MASTER EN SCIENCES INFORMATIQUES

Analyse de l'historique d'évolution d'un système d'information : focus sur les aspects collaboratifs

George, Patrick

Award date:
2017

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2016–2017

Analyse de l'historique d'évolution d'un système d'information : focus sur les aspects collaboratifs

Patrick GEORGE

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Anthony CLEVE

Co-promoteur : Loup MEURICE

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

PATRICK GEORGE

Université de Namur
Faculté d'informatique
Master en sciences informatiques à horaire décalé

Année académique 2016-2017

Analyse de l'historique d'évolution d'un système d'information : focus sur les aspects collaboratifs

Exploiter les données historiques d'un système d'information pour révéler les liens entre les développeurs.

Mots clefs : Co-évolution, comportements sociaux, base de données, données historiques, analyse des réseaux sociaux.

Résumé : L'évolution d'un système informatique est souvent inévitable, qu'il s'agisse d'ajouter une nouvelle fonctionnalité pour répondre à un nouveau besoin, corriger un défaut ou encore en améliorer les performances. L'élément humain est à prendre en compte lors de la maintenance d'un logiciel. En effet, plusieurs développeurs peuvent travailler en équipe pour réaliser l'implémentation d'une tâche, qu'elle soit l'implémentation d'une nouvelle fonctionnalité ou la correction d'un bug. Ces développeurs doivent communiquer efficacement pour se coordonner. Ce mémoire se propose d'explorer l'utilisation de techniques de détection et d'analyse de réseaux sociaux, comme support à la maintenance de la base de données d'un système d'information manipulant intensivement des données. Ces techniques ont pour but d'améliorer les comportements de collaboration et de communication d'une équipe et leurs influences sur la qualité de la maintenance d'un système d'information.

PATRICK GEORGE

University of Namur
Faculty of Compute Science
Master of Science in Compute Science (shift schedule)

Academic Year 2016-2017

Analysis of the evolution of information system : focus on collaborative aspects
Leveraging historical data from an information system to reveal links between developers.

Keywords : Co-evolution, social behavior, database, historical data, social networks analysis.

Abstract : The evolution of a computer system is often inevitable, whether it is adding a new functionality to meet a new need, correcting a fault or improving performance. The human element must be taken into account when maintaining a software. Indeed, several developers can work as a team to implement a task, whether it is the implementation of a new functionality or the correction of a bug. These developers need to communicate effectively to coordinate. This Master thesis proposes to explore the use of techniques for the detection and analysis of social networks, as a support for the maintenance of the database of an information system that intensively manipulates data. These techniques are intended to improve collaborative and communication behaviors of a team and their influence on the quality of maintenance of an information system.

Remerciements

Qu'il me soit donné ici de remercier tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce mémoire.

Mes remerciements tout particuliers sont adressés à Anthony Cleve, promoteur de ce mémoire, et à Loup Meurice, qui en fut le co-promoteur, pour leurs conseils, leur soutien, leur aide et leur enthousiasme si précieux.

Je remercie mes amis et collègues pour leur soutien et leur motivation, m'ayant permis de surmonter les moments difficiles.

Je tiens également à remercier Catherine, Cécile, Denis et Morgan pour leur relecture attentive et leurs remarques judicieuses.

Enfin, merci Larissa, pour ton soutien, ta patience, ton réconfort, de m'avoir permis d'aller au bout de ce projet.

Table des matières

Liste des acronymes

CRM Customer Relationship Management. 5

CSBFs Cross-System-Bug-Fixings. 21

CSV comma-separated values. 31

DAHLIA **DA**tabase sc**H**ema evo**L**ut**I**on **A**nalysis. 7, 30, 32, 33, 65

DDL Data Definition Language. 8, 9

DISS Data-Intensive Software Systems. iv, 5, 7, 11, 22, 26, 27, 30–32, 38, 66, 81, 87, 90, 92

DML Data Manipulation Language. 8

DRE Database Reverse Engineering. 8, 9

EMR Electronic Medical Record. 5, 31

ERP Enterprise Ressource Planning. 5

MSR Mining Software Repositories. 9, 11, 21, 25, 32, 82

ORM Object-Relational-Mapping. 35, 37

SGBD système de gestion de base de données. 5, 6

SI Système d'information. 4, 8, 11, 25, 29, 38, 88, 92

SNA Social Network Analysis. 12, 25, 27, 66

SQL Structured Query Language. 8, 9, 32

SVN Subversion. 7

Index

Introduction

L'évolution d'un système informatique est souvent inévitable, qu'il s'agisse d'ajouter une nouvelle fonctionnalité pour répondre à un nouveau besoin, corriger un défaut ou encore en améliorer les performances. [Brooks 1995] affirme déjà, en 1975, que le coût de cette maintenance peut représenter, durant toute la durée de vie d'un programme, 40% ou plus du coût de son développement.

Depuis, la technologie peut avoir changé mais l'élément humain des projets informatiques est resté identique. Cet élément humain est donc à prendre en compte. En effet, lors de la maintenance d'un logiciel, plusieurs développeurs peuvent travailler en équipe pour réaliser l'implémentation d'une tâche, qu'elle soit l'implémentation d'une nouvelle fonctionnalité ou la correction d'un bug. Ces développeurs doivent communiquer efficacement pour se coordonner.

Le succès de la gestion d'un projet est hautement dépendant d'une communication organisationnelle efficace[guide 2013]. Un chef de projet peut suspecter que l'échec d'une étape du projet soit due à un manque de communication. Il peut donc éventuellement souhaiter d'objectiver les problèmes de collaboration et de communication dans son équipe. Ceci permettrait d'ajuster les comportements des développeurs, d'assurer le succès du projet et d'en maîtriser les coûts.

De nombreuses recherches se sont focalisées sur les aspects sociaux du développement d'un projet informatique. De ces recherches, ont abouti divers méthodes et outils permettant d'établir des liens entre les développeurs en exploitant les données contenues dans les dépôts de versions. De tels liens sont établis soit au travers des archives de communication soit en utilisant l'historique des modifications apportées aux artefacts techniques.

Les liens entre développeurs construisent alors des réseaux sociaux. Ces réseaux peuvent être analysés qualitativement, c'est-à-dire qu'ils peuvent être visualisés pour évaluer la structure organisationnelle d'un projet. Ils peuvent aussi être analysés quantitativement, c'est-à-dire que des indicateurs peuvent caractériser numériquement les composants de ce réseau dans le but d'obtenir des statistiques.

Rétrospectivement, il serait possible de corréler des patterns organisationnels avec d'autres informations comme le nombre de bugs ouverts et leur séverité, l'échec ou le succès d'un effort de développement. Une telle corrélation n'est pas nécessai-

rement la cause du succès ou de l'échec et donc doit être évaluée avec un regard critique.

De tels outils peuvent donc fournir des informations intéressantes pour gérer une équipe.

Pourtant, lorsqu'il s'agit d'étudier l'évolution et la maintenance du schéma d'une base de données et du code y accédant, les travaux de recherche se font rare. [Mens et Goeminne 2011] ont déjà étudié les aspects sociaux, mais se sont focalisé sur le code accédant aux bases de données. Nous déplorons l'absence d'usage des outils d'analyse des réseaux sociaux dans le domaine de la maintenance du schéma d'une base de données et du code y accédant.

Nous pensons que de telles méthodes pourraient apporter une réelle plus-value à la maintenance de logiciels ayant un lien étroit avec une base de données, dont le schéma est large, les données nombreuses et continuellement manipulées.

Dans ce mémoire, nous ferons l'état de l'art des recherches qui se sont focalisées sur les aspects sociaux du développement d'un projet informatique. Nous poserons ensuite des questions de recherche précises. Ces questions viseront à appliquer les méthodes existantes de détection et d'analyse des réseaux sociaux aux artefacts techniques tels que le schéma de la base de données et le code y accédant.

Pour terminer, notre contribution consistera en l'élaboration d'un prototype permettant de construire des réseaux sociaux à partir des changements apportés au schéma de la base de données. Nous évaluerons notre méthode sur les données historiques de deux projets réels et nous proposerons des pistes de travaux futurs.

Le premier chapitre de ce travail introduit les concepts de base et dresse un état de l'art des méthodes de détection et d'analyse des réseaux sociaux. Le second chapitre identifie nos questions de recherche et évalue les données historiques disponibles. Enfin, le troisième chapitre présente des méthodes de résolution des questions de recherche et les résultats obtenus par notre prototype.

Chapitre 1

État de l'art

1.1 Introduction

Le présent chapitre vise à expliciter les notions sur lesquelles nous allons travailler. Nous présenterons d'abord le type de système d'informations que nous allons étudier ainsi que les aspects liés au développement de ces systèmes. Nous nous intéresserons ensuite, plus particulièrement, aux aspects collaboratifs du développement de ces systèmes, qui peuvent s'avérer très intéressants dans le cadre de la rétro-ingénierie. A cet effet, nous établirons une revue de littérature permettant de mieux cerner les aspects collaboratifs qui ont déjà été observés, ainsi que méthodes employées pour y parvenir.

1.2 Concepts de base

1.2.1 Les systèmes d'information

D'un point de vue général, un système d'information permet d'organiser des données et de les présenter sous forme d'informations à un utilisateur pour lui apporter des réponses ou pour résoudre des problèmes. Il est défini comme "un ensemble d'éléments (personnel, matériel, logiciel...) permettant d'acquérir , traiter, mémoriser et communiquer des informations"[*Chapitre 1 : Système d'information de l'entreprise*]. Les composants informatiques matériel et logiciel d'un Système d'information (SI) ne sont qu'un sous-ensemble d'un SI. Dans ce travail nous sommes intéressés par la collaboration entre les développeurs lors du développement de la composante logicielle. Un exemple simple de composante logicielle d'un SI est la feuille de calcul électronique établie au moyen d'un tableur. Elle organise les données contenues dans la feuille sous forme d'informations intelligibles pour l'utilisateur, comme un calcul de moyenne, une courbe ou un graphique. Un exemple plus complexe de composante logicielle d'un SI est le cas d'une application manipulant une large quantité

de données, stockant celles-ci dans une base de données, manipulant ces données et les présentant sous forme d'information. C'est sur ce type d'application que sera portera notre attention.

1.2.2 Les *Data-Intensive Software Systems (DISS)*

Il existe plusieurs appellations pour décrire les systèmes informatiques dont l'artefact base de données est primordial et qui manipulent une quantitée importante de données. Ces sytèmes sont appellés *database-intensive system*, *database-intensive application*, *data-intensive application* ou *data-intensive software systems (DISS)*.

Pour notre part, nous avons choisi de nous référer à la dénomination la plus récente qui est DISS.

Les DISS sont décrits par [Goeminne, Decan et Mens 2014] comme des logiciels manipulant intensivement des données, composés d'une base de données et du code source qui implémente les fonctionnalités principales du système. [Cleve, Mens et Hainaut 2010] mentionnent aussi la forte interraction entre le programme qui implémente la logique business et le système de données qui contient les objets business.

Les DISS peuvent prendre la forme de dossier médicaux électronique (DME) [1], de logiciels de gestion de la relation clients (GRC) [2], de logiciels Enterprise Ressource Planning (ERP), etc.

Ces logiciels comprennent un artefact comprenant une base de données et un système de gestion de base de données (SGBD). Ceci permet d'assurer la persistance des données et de manipuler celles-ci. De telles bases de données sont, à l'heure actuelle, le plus souvent du type relationnel. En effet, selon le classement [*DB-Engines Ranking*] illustré en figure 1.1 parmi les 10 SGBD les plus populaires en mars 2017, les quatre premiers : Oracle, MySQL, Microsoft SQL server, PostgreSQL gèrent des bases de données relationnelles et 7 sur 10 sont de type relationnel.

1.2.3 Une évolution constante

Dans ce mémoire, nous allons analyser l'historique d'évolution des DISS. Les DISS, comme beaucoup de logiciels, sont amenés à évoluer constamment au cours de leur développement et de leur exploitation. Les raisons de cette évolution constante sont les suivantes :

— Pendant la phase de développement, les méthodes de gestion de projets informatiques du type agile (Scrum, XP, etc.) impliquent des cycles de développement itératif. Les méthodes agiles remplacent la méthodologie Waterfall

1. Traduction de l'anglais *Electronic Medical Record (EMR)*
2. Traduction de l'anglais *Customer Relationship Management (CRM)*

322 systems in ranking, March 2017

Rank Mar 2017	Rank Feb 2017	Rank Mar 2016	DBMS	Database Model	Score Mar 2017	Score Feb 2017	Score Mar 2016
1.	1.	1.	Oracle	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis	Key-value store	113.01	-1.03	+6.79

FIGURE 1.1 – Classement des gestionnaires de bases de données selon leur popularité [*DB-Engines Ranking*]

quand les exigeances ne sont pas claires dés le départ et/ou sont évolutives. Elles permettent de capturer de nouvelles exigeances tout au long du projet lors de chaque itération :

— lors de l'exploitation du système, des changements peuvent subvenir visant à corriger des erreurs résiduelles dans le logiciel ;

— de nouveaux besoins peuvent aussi amener à ajouter de nouvelles fonctionnalités et à initier un nouveau cycle de développement ;

— parfois, pour des raisons d'optimisation, il faut retravailler le code sans pour autant y ajouter des fonctionnalités ni en corriger les erreurs. Cette étape d'optimisation est appellée en anglais "code refactoring".

On peut aussi catégoriser deux types différents d'évolution des systèmes en fonction du type d'exigences [CLEVE, MENS et HAINAUT 2010] :

— **L'évolution fonctionnelle des systèmes**, qui est initiée par des exigences fonctionnelles, c'est-à-dire qui visent à changer la fonction du système. Par exemple l'ajout, la modification ou la suppresion de fonctionnalités.

— **L'évolution non-fonctionnelle des systèmes**, qui est initiée par des exigences non-fonctionelles, c'est-à-dire qui visent à changer la structure du système tout en préservant sa fonction. Par exemple, on parlera d'évolution non-fonctionnelle lorsqu'on procède à la migration d'un SGBD à un autre tout en préservant les fonctionalités du système.

Au cours de l'évolution du logiciel, chaque modification apportée engendre une nouvelle version de celui-ci. Selon le type de changement, on peut distinguer des versions majeures (nouvelles fonctionalités, refactorisation) et des versions mineures (fonctionnalités secondaires, correction d'erreurs).

1.2.4 Logiciels de gestion de versions

Lorsque plusieurs développeurs travaillent sur un même projet, la gestion du code et la préservation de la cohérence de celui-ci deviennent très rapidement impossible lorsqu'elles sont effectuées manuellement. Il existe des outils pour faciliter la gestion du code tout au long des versions successives. Au moyen de ces outils, les développeurs collaborant sur un même projet peuvent soumettre leurs changements sur un dépôt distant. Ces dépôts peuvent être hébergés sur Internet ou sur un réseau local. Ces outils qui gèrent de tels dépôts sont des logiciels de gestion de versions[*What is version control | Atlassian Git Tutorial*]. Il existe de nombreux logiciels de gestion de versions. Cependant, les deux systèmes les plus populaires actuellement sont GIT et Subversion (SVN)[*Version Control Systems Popularity in 2016* 2016].

1.2.5 Impact de l'évolution sur le schéma et le code

Dans le cas des DISS, des changements peuvent nécessiter d'altérer le schéma de la base de données pour permettre la prise en charge de nouvelles données non prises en compte lors d'une précédente version.

Pour garantir le bon fonctionnement du logiciel,[Cleve et Hainaut 2006] ont insisté sur la nécessité de préserver la cohérence entre les trois éléments suivants : le schéma de la base de données, les programmes qui accèdent au schéma et les données existantes. Pour faciliter la maintenance du système, [Cleve, Mens et Hainaut 2010] ont tenu compte d'un quatrième élément : le modèle de conception auquel le programme doit se conformer.

1.2.6 Co-évolution et Co-transformation

La **co-évolution** est l'évolution simultanée de plusieurs éléments interdépendants. Pour assurer le bon fonctionnement des DISS, la co-évolution entre les quatre éléments précités au point 1.2.5 est indispensable.

Pour simplifier la co-évolution entre le code source et le schéma de la base de données, [Meurice et Cleve 2016] ont développé **DA**tabase sc**H**ema evo**L**ut**I**on **A**nalysis (DAHLIA). DAHLIA est un outil de visualisation des liens entre le code et la base de données, ce qui permet de soutenir les efforts de co-évolution des DISS. Cet outil est appliqué aux systèmes Java, dont la base de données est accessible au moyen de différentes technologies qui sont parfois co-existantes dans les projets (e.g. JDBC, Hibernate, JPA).

La **co-transformation** est une manière automatisée d'assurer la co-évolution. L'outil de co-transformation utilise alors les liens partagés entre les artecfacts qui doivent co-évoluer entre une situation de départ et une situation attendue.

A cet effet, [Cleve et Hainaut 2006] proposent un prototype d'outil de co-transformation avec lequel la transformation du programme se fait automatiquement en se basant sur une transformation du schéma. Cet outil de co-transformation a été conçu pour permettre le passage d'un Data Manipulation Language (DML) à un autre (ici COBOL vers Structured Query Language (SQL)). Leur approche a été experimentée sur des programmes de petite à moyenne taille.

1.2.7 Rétro-ingénierie

La rétro-ingénierie dans le sens large du terme est définie comme "Le processus d'analyse d'un certain système pour créer des représentations du système à un niveau d'abstraction plus élevé."[CanforaHarman et Di Penta 2007].

Dans ce travail, nous allons utiliser une méthode de **rétro-ingénierie logicielle** : l'analyse historique d'un SI.

De manière générale, la rétro-ingénierie logicielle est utile quand :

— La documentation est tout simplement absente ;

— La documentation est obsolète ou incomplète [Briand 2003] bien qu'elle soit exigée par les standards de dévelopements ;

— Il faut continuer à développer un projet open source où la documentation est négligée ou de qualité variable[Gacek et Arief 2004].

La retro-ingénierie d'un SI consiste à dériver de l'information à partir d'artefacts logiciels (e.g. son code) et à la transformer en une représentation abstraite plus compréhensible pour un humain, parfois de manière automatisée pour en maximiser les bénéfices[CanforaHarman et Di Penta 2007].

Tout logiciel est susceptible d'évoluer quelle qu'en soit la raison, e.g. en fonction de l'évolution des besoins des utilisateurs. Dans le cas des logiciels comprenant un artefact qui est une base de données, cela nécessite une documentation à jour de la conception de celle-ci.

La **rétro-ingénierie des bases de données** ou en anglais Database Reverse Engineering (DRE) vise donc à reconstruire une documentation à jour de la base de données (i.e. les schémas conceptuel, logique, physique ainsi que le code SQL). Ce processus exploite les quatre sources d'information suivantes [Hainaut 2009],[Cleve et al. 2015] :

1. le code Data Definition Language (DDL) ;
2. les programmes d'applications ;
3. les données stockées ;
4. l'historique d'évolution.

Le code DDL

Le code SQL - DDL est utilisé pour créer les objets de la base de données (les tables et les colonnes), les contraintes et les clefs étrangères. Ceci peut être utile pour reconstruire des connaissances sur la base de données.

Dans la pratique, cette source d'information peut s'avérer insuffisante car :

— le schéma peut contenir des sections peu explicites [Hainaut 2009] ;

— le schéma peut contenir des structures obsolètes au fur et à mesure de son évolution (*dead schema*) [Cleve et al. 2015] ;

— le schéma ne contient pas de clefs étrangères qui peuvent établir un lien explicite entre les tables [Cleve et al. 2015].

Les programmes d'applications

Le code source des programmes d'applications peut apporter des informations complémentaires sur les structures et les propriétés des données.

Dans la pratique, cette source d'information n'est pas toujours suffisante. Par exemple, si pour une même version d'un logiciel, le schéma de la base de données ne correspond pas totalement avec le code source du programme (Dead code ou Dead schema).

Les données stockées

Pouvoir analyser les données d'un système en production peut permettre, lors du processus de DRE, de recouvrir des relations entre objets qui ne sont pas explicites dans le code SQL - DDL.

Dans la pratique, cette source d'information n'est pas toujours disponible pour faciliter le processus de DRE en raison de la nature sensible de certaines informations. Une solution serait de rendre anonymes les données si c'est possible dans le cadre défini.

L'historique d'évolution

L'analyse de l'historique d'évolution a gagné en importance dans le processus de rétro-ingénierie logicielle de manière générale. L'analyse historique d'évolution des bases de données proposée par [Cleve et al. 2015] est aussi utilisée lors du processus de DRE.

L'analyse de l'historique d'évolution se base sur les données historiques extraites par les techniques de Mining Software Repositories (MSR)[Meurice et Cleve 2014]. Le MSR étudie les données disponibles dans les dépôts logiciels tels que les

systèmes de contrôle de version, issue/bug-tracking systems, ou archives de communication.

1.3 Revue de la littérature

Dans cette section, nous tenterons d'élaborer un état de l'art des travaux existants dans les domaines de la détection des liens entre développeurs, au moyen de techniques de MSR. Premièrement, nous ferons l'état des lieux des différentes approches de détection et d'analyse de ses liens. Ensuite, nous discuterons des résultats existant dans le domaine des SI. Finalement, dans le cas des DISS, nous soulignerons le manque d'application de ces méthodes quand il s'agit d'utiliser le schéma de la base de données et le code y accédant pour établir des liens entre développeurs.

1.3.1 Détection des aspects collaboratifs d'un projet informatique

[Wolf et al. 2009] ont dressé un bref état de l'art concernant les méthodes de construction de réseaux de développeurs. Ils décrivent trois catégories selon l'information utilisée pour connecter deux développeurs dans un réseau.

1. **Technical based** Lorsque deux développeurs changent un même fichier ou module.
2. **Project-wide communication** Lorsque deux développeurs ont utilisé un canal de communication lié au projet.
3. **Task-related communication** Lorsque deux développeurs ont communiqué ensemble au sujet d'une tâche en particulier, comme implémenter une fonctionnalité ou corriger un bug.

Le résultat de chaque méthode produit un ou plusieurs graphe(s) représentant un ou plusieurs réseau(x) de développeurs. Ces réseaux sont visualisables et analysables dans le but de mieux comprendre les relations entre les développeurs.

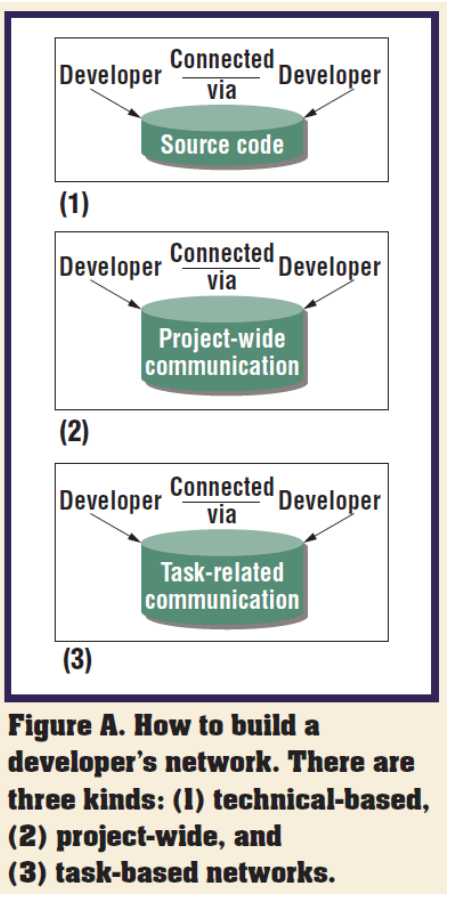


Figure A. How to build a developer's network. There are three kinds: (I) technical-based, (2) project-wide, and (3) task-based networks.

Figure 1.2 – Méthodes de construction d'un réseau de développeurs selon [**Figure A** ; Wolf et al. 2009]

Approches basées sur les artefacts techniques

[De Souza, Froehlich et Dourish 2005] arrivent à détecter des comportements sociaux en utilisant tout d'abord une méthode de **call-graph analysis** pour

établir un lien entre les développeurs quand **les méthodes dont ils sont les auteurs s'appellent les unes les autres** (analyse statique). Ensuite, ils utilisent une méthode d'analyse de ces réseaux construits au moyen des contributions qu'apportent des développeurs à un ou plusieurs modules communs. En analysant plusieurs projets open-source, ils identifient différentes formes de participation illustrées en figure 1.3. Pour terminer, ils analysent aussi l'évolution du pattern de participation au fil des versions du projet.

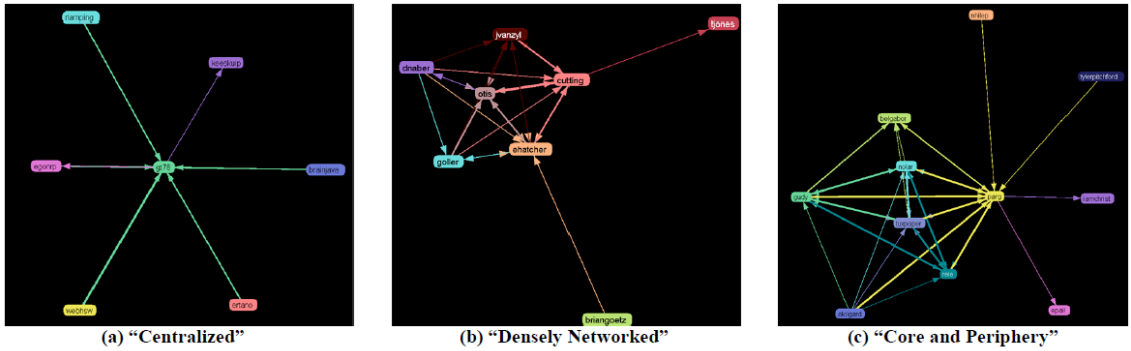


Figure 2: Forms of participation

FIGURE 1.3 – Différentes formes de participations obtenues par [**figure 2**; DE SOUZA, FROEHLICH et DOURISH 2005]

[MARTINEZ-ROMO et al. 2008] font la différence entre les logiciels développés dans l'industrie et les logiciels libres et open source. Ils postulent que, dans le privé, les développeurs sont organisés formellement et travaillent en équipes auxquelles sont assignés certains modules du système. Ceci n'est pas le cas dans les logiciels libres et open-source, où la structure est moins formelle et où les développeurs ont accès à l'entièreté du projet et peuvent collaborer sur n'importe quel module selon les règles convenues entre les développeurs. Comme il n'y a pas de gestion centrale, les réseaux d'intéractions résultants peuvent être complexes. Tout d'abord, ils construisent un réseau social de développeurs dont ils sont les noeuds. Les arrêtes lient deux développeurs s'ils ont contribué sur au moins **un fichier en commun, ou n'importe quel fichier dans un répertoire commun**. Les arrêtes sont pondérées. Le poids de l'arrête représente le nombre de commits sur des fichiers communs. Ensuite, ils analysent le réseau obtenu avec des méthodes de Social Network Analysis (SNA) pour caractériser le réseau et ses noeuds. Ils étudient l'influence que peut avoir la collaboration entre une industrie et une communauté de développeurs de logiciels libres sur les résultats obtenus. Cette influence est observée via l'analyse du réseau social au fil de l'historique des versions.

Par exemple, une des métriques obtenue par leur analyse du réseau social est le **degré de coordination**. Le degré de coordination d'un noeud dans un réseau social est sa capacité à échanger de l'information.

Ils définissent le degré de coordination γ_{ij} entre deux noeuds i et j comme $\gamma_{ij} = e^{-\varepsilon d_{ij}}$, où $d_i j$ est la distance entre deux noeuds et ε est un réel constant, et représente la force de la relation qu'ils appellent **force de coordination**. Le **degré de coordination total** Γ_i d'un noeud i est la somme de tous les degrés de coordination entre ce noeud et les autres $\Gamma_i = \Sigma_{j=1}^{N} \gamma_{ij}$ où N est le nombre total de noeuds du graphe. Le degré total de coordination est somme des informations appartenant au réseau que ce noeud est capable de recevoir.

Un exemple de résultat en figure 1.4 représente en abscisse les différentes versions et en ordonnée le degré de coordination des 40 développeurs qui soumettent le plus de contenu. Au dessus du graphe, on retrouve les étapes principales du projet qui correspondent à l'influence d'une société privée sur le degré de coordination des développeurs et les distributions du logiciel.

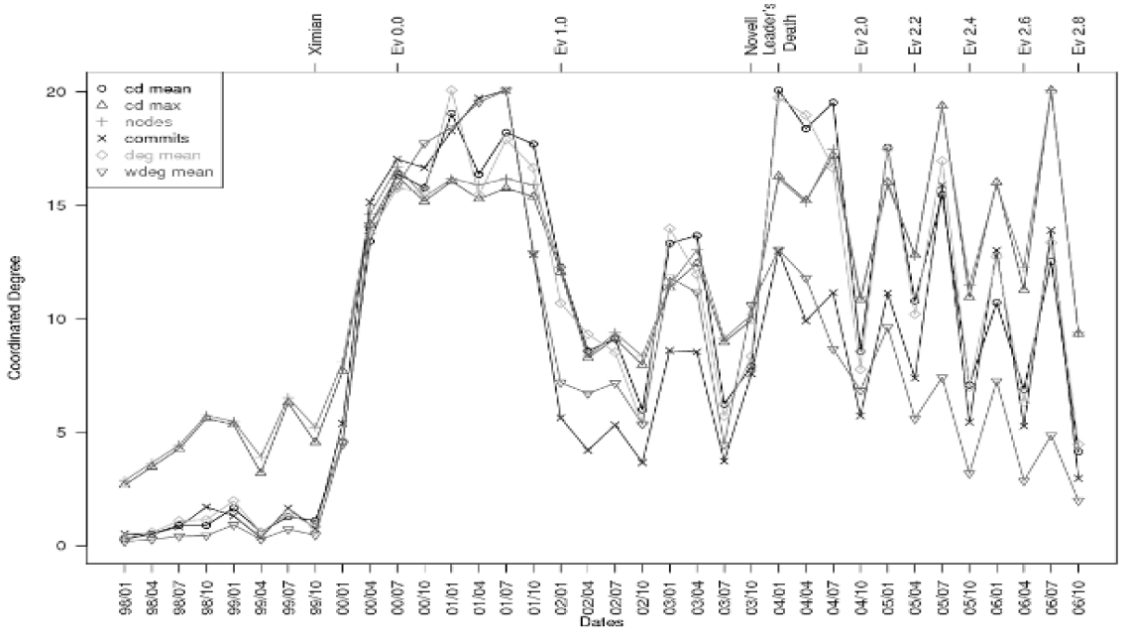


FIGURE 1.4 – Degré de coordination pour les 40 développeurs qui soumettent le plus de contenu durant l'évolution [**Figure 1b** ; MARTINEZ-ROMO et al. 2008]

On constate à l'aide d'un tel schéma, qu'une entreprise peut avoir une influence positive sur le degré de coordination entre les développeurs. On constate aussi que le degré de coordination augmente avant la publication d'une nouvelle version. Le départ d'un membre de l'équipe, dans cet exemple la mort du chef developpeur, peut avoir comme effet de diminuer temporairement le degré de coordination entre les développeurs.

[VALETTO et al. 2007] comparent la structure en réseau de l'organisation des développeurs collaborant sur un projet et la structure en réseau du logiciel. Ils analysent si les structures de l'organisation et les structures du logiciel sont congruentes, dans le but de déterminer le degré d'alignement entre les relations sociales et les re-

lations entre les éléments du logiciel.

Un exemple de réseau socio-technologique est illustré en figure 1.5

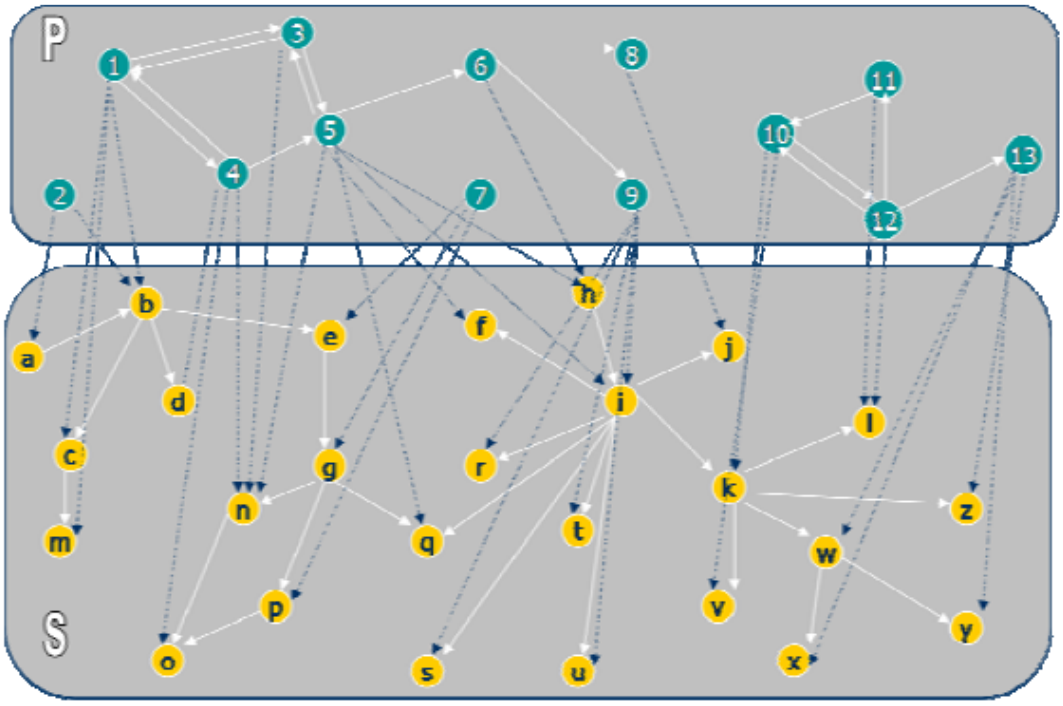


FIGURE 1.5 – Réseau socio-technologique [**Figure 1** ; VALETTO et al. 2007]

Les relations entre les noeuds des développeurs de l'ensemble P peuvent être obtenues en **observant les communications entre les développeurs**. Dans le cas où il n'y a pas d'enregistrements des communications, cette information peut être estimée si les développeurs **ont travaillé sur un même artefact**. Les relations entre les noeuds des artefacts peuvent être élaborées avec une analyse statique qui construirait l'arbre d'appel des méthodes du programme. Les relations entre l'ensemble P et S sont établies si un développeur a travaillé sur un artefact pendant la durée du projet.

Cette technique est applicable quand on cherche à savoir si la structure de l'organisation correspond à la sous-division des responsabilités dans un projet. Ceci peut permettre de gérer plusieurs équipes responsables de modules et de s'assurer que l'exécution des modules correspond à la distribution.

Approche basée sur les communications entre développeurs

[BIRD et al. 2008] utilisent l'extraction et l'étude **des emails partagés entre les développeurs** de projets open source afin de déduire la structure en réseau de l'organisation de ces projets. Ils utilisent la fréquence de communication (le poids des arrêtes) pour déterminer des sous-groupes dans la communauté du projet. Ce-

pendant, ils n'utilisent pas le code source pour détecter les collaborations entre les développeurs.

Un exemple de réseau obtenu est illustré en figure 1.6. Les noeuds en ellipse y représentent les participants du projet. Les noeuds en losange représentent les développeurs. Les poids des arrêtes, qui sont fonction du nombre de messages entre participants, ne sont pas représentés sur le graphe mais ont été utilisés pour grouper les participants et les développeurs en sous-communautés.

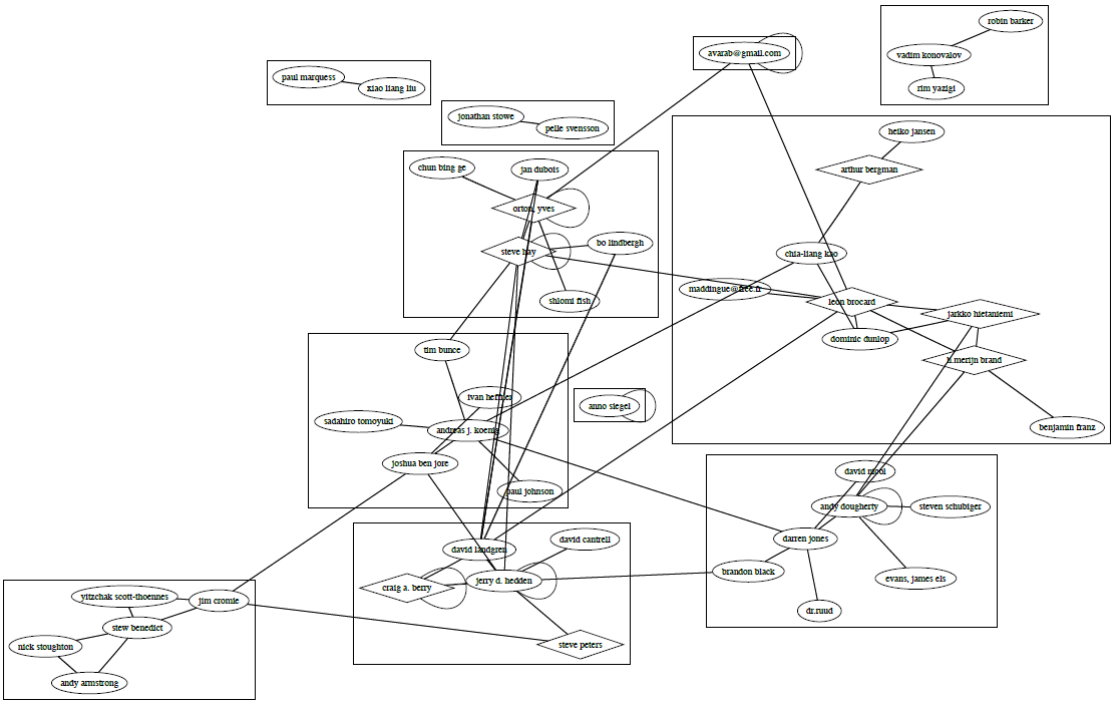


FIGURE 1.6 – Structure de la communauté de Perl d'avril à juin 2007.[**Figure 3** ; BIRD et al. 2008]

Approches basées sur les tâches entre développeurs

Certains outils favorisent une bonne collaboration entre développeurs (e.g. IBM rational team concert). Dans ce cas, cet outil collaboratif est utilisé par l'équipe de développeurs pour gérer tous les aspects de leur travail, tels que le planning des itérations et des distributions, la gestion des modifications, le suivi des défauts, le contrôle des sources et l'automatisation des **builds**[*IBM Rational Team Concert*]. Les données sauvegardées sur le dépôt de version sont alors riches d'informations sur les tâches auxquelles différents développeurs sont assignés. [WOLF et al. 2009] proposent une méthode générique applicable quand le dépôt de version contient les informations sur les développeurs, les tâches, les communications et les artefacts techniques. Pour cela, ils utilisent trois sources d'informations : les membres du projet, les tâches collaboratives et **les communications liées aux tâches** pour

construire un réseau social qui est le reflet de la collaboration entre les développeurs lors d'un build. Ceci est intéressant pour comprendre si un build défectueux est par exemple dû à un manque de communication. Leur approche permet de filtrer très finement le réseau social autour d'une ou plusieurs tâches collaboratives comme l'implémentation d'une fonctionnalité ou la correction d'un bug. Un manque de communication entre plusieurs responsables d'une tâche pendant un build peut ainsi prédire un potentiel échec de ce build.

Le **build1** de la figure 1.7 est connu comme défectueux. Pour construire son réseau, il faut filtrer les **contributors** entre $t0$ et $t1$ et identifier les tâches **work item** associées avec les changements **change set** apportés par le build1. Les arrêtes et leur poids sont déterminés par le nombre de commentaires sur une tâche commune à deux développeurs. La même méthodologie est appliquée pour contruire le réseau du **build2** qui, lui, est fonctionnel. L'analyse comparative des deux réseaux peut donc permettre de mieux comprendre les raisons de l'échec d'un build s'il est dû ou non à un manque de communication entre les développeurs.

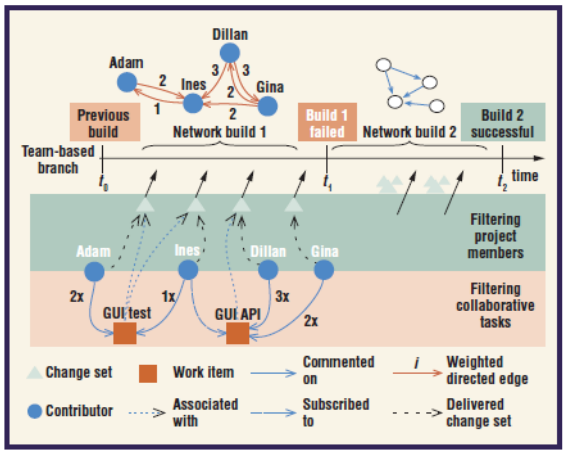


FIGURE 1.7 – Construction de réseaux sociaux basé sur les tâches pour prédire l'échec d'un build.[**Figure 2** ; WOLF et al. 2009]

Un autre type de source d'informations pour détecter des relations entre développeurs en se basant sur une tâche à accomplir est le système de suivi de bug. [SUREKA, GOYAL et RASTOGI 2011] utilisent ces informations pour construire des réseaux de développeurs selon le niveau de gravité des bugs sur lesquels ils travaillent. De tels réseaux permettent d'identifier les développeurs qui travaillent sur des bugs d'un même niveau de gravité, ou ceux qui travaillent sur des bugs de plusieurs niveaux de gravité.

Un exemple de graphe obtenu est représenté en figure 1.8. Leur analyse permet dès lors d'identifier les experts ou les développeurs dont les connaissances sont exclusives, ce qui sera utile pour gérer les risques et comprendre la structure de l'organisation.

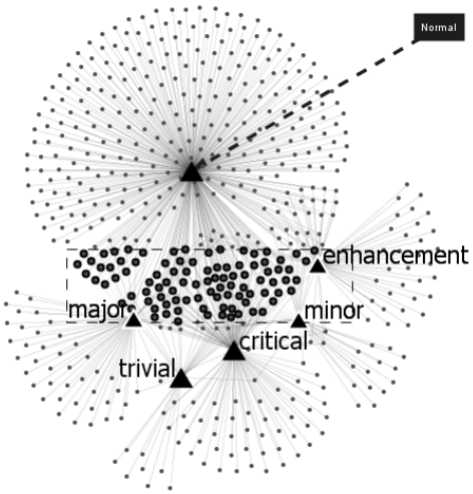


Figure 4: A two mode network depicting relationship between bug severity and developers

FIGURE 1.8 – Réseau de relations entre les développeurs et la gravité des bugs sur lesquels ils travaillent.[**Figure 4** ; SUREKA, GOYAL et RASTOGI 2011]

L'outil Tesseract [SARMA et al. 2009] permet de visualiser simultanément les relations sociales et techniques en extrayant les informations d'un dépôt de version. Il met également en évidence le lien ou l'absence de lien entre les relations sociales et techniques. Il offre en outre, la possibilité d'explorer interactivement ces relations et d'en observer les changements au fil du temps.

Un exemple de visualisation obtenue avec l'outil est illustré en figure 1.9. La situation au moment (a) et (b) reflète des pics d'activités dans le projet. **La situation en (a)** correspond au premier pic d'activité où Stephen Walther est le contributeur principal (il a d'ailleurs apporté des changements à tous les fichiers). Dans le graphe de droite, on constate que Stephen communique bien avec les autres développeurs (les arrêtes vertes), mais que les autres développeurs ne communiquent pas beaucoup entre eux (les arrêtes rouges). Le réseau de fichiers à gauche est très connecté, et la période observée est marquée par une liste de bugs continuellement croissante (courbes en bas).

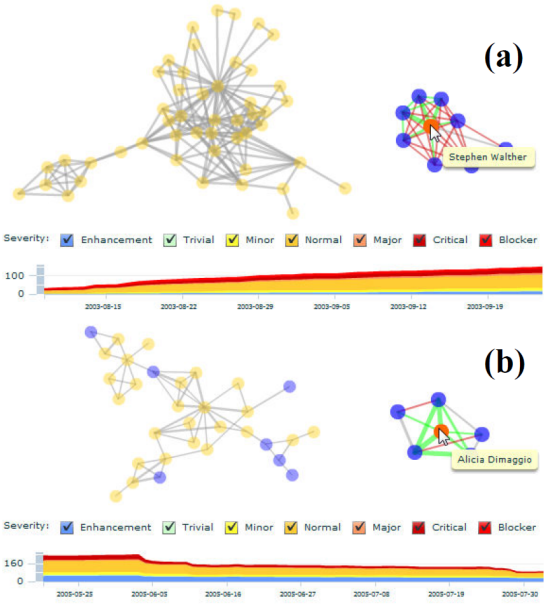


Figure 2. Contrasting development patterns.

FIGURE 1.9 – Contraste entre deux patterns de développement obtenus par l'outil Tesseract[**Figure 2** ; SARMA et al. 2009]

La situation en (b) montre un pic d'activité ultérieur. Alicia Dimaggio est le prin-

cipal contributeur. Dans le graphe de droite, on constate qu'elle communique avec les autres contributeurs, mais que dans ce cas, le niveau de communication est également suffisant entre les autres contributeurs. Le réseau de fichiers à gauche est moins dense et la période observée est marquée par une diminution de la liste des bugs (courbes en bas). Les auteurs n'ont pas prouvé de liens de causalité entre les patterns de communications et la densité du réseau de fichiers impacté ni la liste des bugs. Mais cet outil produit certainement des informations intéressantes sur l'état d'un projet.

Approches basées sur les informations de plusieurs projets

L'outil Graphmania [Ohira et al. 2005] permet de mettre en évidence des similarités entre des développeurs et des projets. Pour ce faire, il utilise les différents projets open-source auxquels les développeurs ont participé pour construire trois types des réseaux. Premièrement, un réseau développeurs-projets où le poids des arrêtes correspond au degré de similarité entre les noeuds du graphe (illustré en figure 1.10).

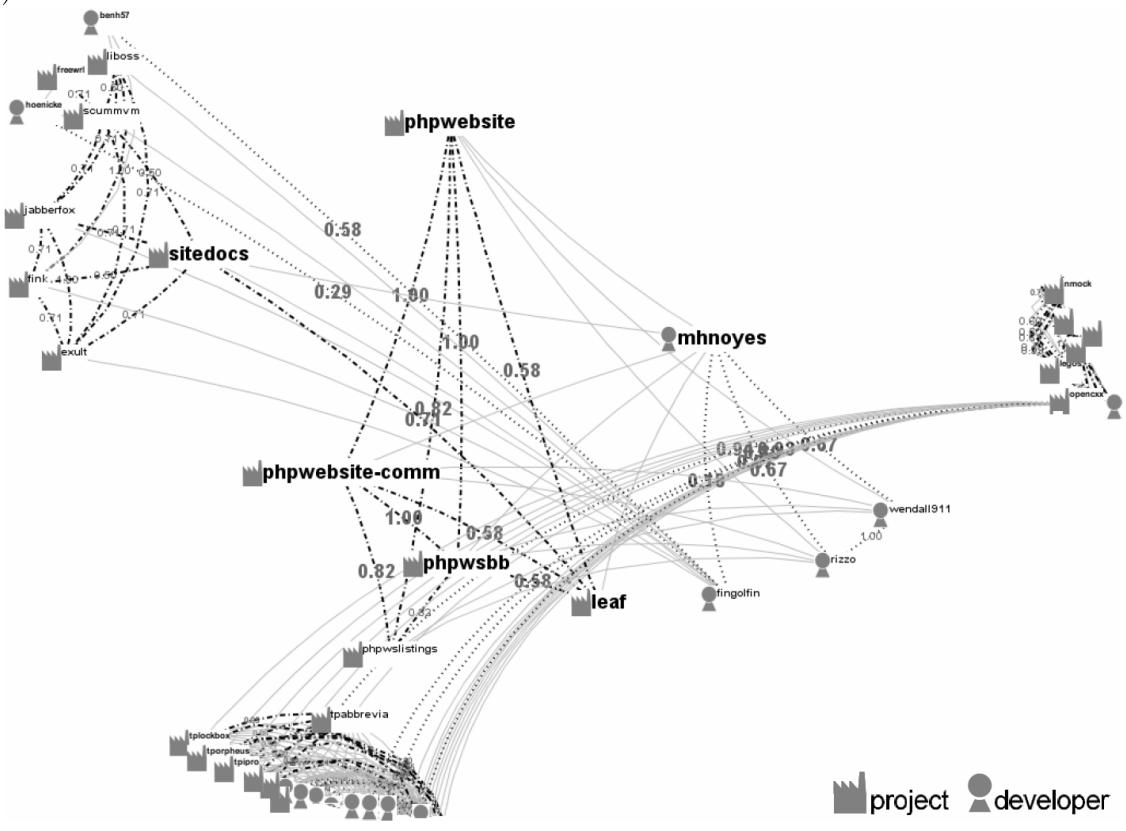


Figure 1.10 – Réseau Developpeurs-Projets [**Figure 1** ; Ohira et al. 2005]

Deuxièmement, si deux développeurs ont participé ensemble sur un même projet plusieurs fois, alors on estime qu'il existe une importante similarité entre eux. Un réseau de développeurs peut alors être construit. On peut identifier, sur un tel schéma, les développeurs qui sont les seuls liens entre deux parties du réseau. Sur le réseau représenté en figure 1.11, on constate que les développeurs **ozmosys** et **om-bill** sont des noeuds qui peuvent totalement déconnecter le graphe s'ils n'étaient plus présents.

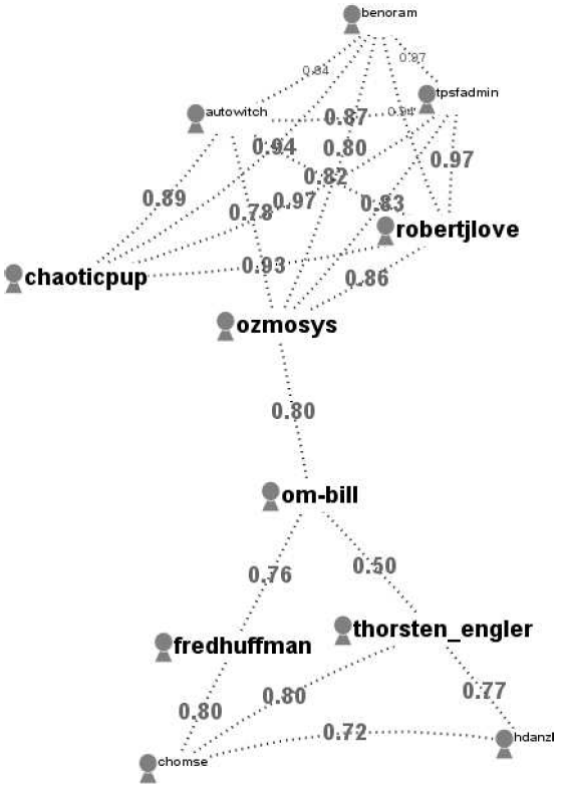


FIGURE 1.11 – Réseau de développeurs [**Figure 2**; OHIRA et al. 2005]

Troisièmement, de la même manière, si les mêmes développeurs travaillent ensemble sur plusieurs projets, alors on considère que la similarité entre ces projets est forte. Un réseau de projets comme représenté en figure 1.12 peut alors être construit.

FIGURE 1.12 – Réseau de projets [**Figure 3** ; OHIRA et al. 2005]

En somme, cet outil est intéressant pour partager l'expertise entre plusieurs projets ou pour trouver un expert dans un projet similaire. Il ne détecte par contre pas de similarité entre des développeurs sur base d'un seul projet.

[VAN ANTWERP et MADEY 2010] utilisent quant à eux les données extraites d'un **réseau de développeur** où les liens entre les développeurs se sont établis sur des projets open sources **populaires sur le long terme** et les **indicateurs d'activité** des développeurs. Ils observent à partir de ces données que les projets open sources les plus populaires ont plus de chances de contenir un réseau de développeurs ayant déja existé dans un projet précédent. Et donc que ces précédents réseaux de collaboration fructueuse pourraient influencer le taux d'activité d'un nouveau projet open sources. Un réseau de collaboration préexistant pourrait ainsi être un prédicteur du succès d'un futur projet open sources.

Dans le cas de logiciels partageant des caractéristiques similaires, [CANFORA et al. 2011] portent une attention particulière à la propagation de la **correction des bugs entre ces systèmes** et les caractéristiques sociales des contributeurs

correspondant sur les mailing lists des projets. Ils appellent ce phénomène Cross-System-Bug-Fixings (CSBFs).

1.3.2 Différents types de participation dans un projet

[Nakakoji et al. 2002] catégorisent huit rôles selon le type de contributions que peut avoir un membre d'un projet :

— les utilisateurs passifs ;

— les lecteurs ;

— les rapporteurs de bugs ;

— les correcteurs de bugs ;

— les développeurs périphériques ;

— les développeurs actifs ;

— les membres principaux ;

— le chef de projet.

[Mens et Goeminne 2011] analysent avec des techniques de MSR les types d'activités que les membres d'un projet peuvent exercer. Ils identifient trois types d'activités au moyen des informations disponibles : la soumission de changements sur le code source, les communications par mail et les changements sur les rapports de bugs. Ils constatent que certains membres peuvent cumuler différents types d'activités comme illustré en figure 1.13.

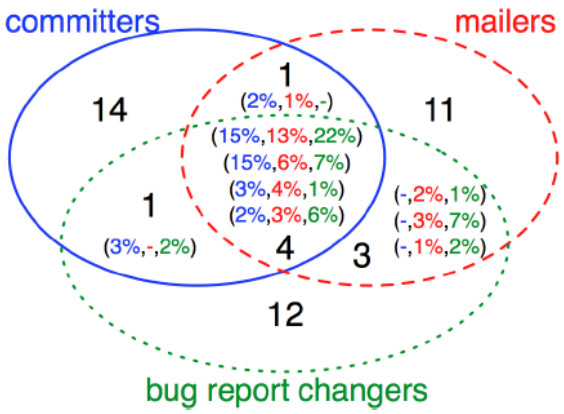


Figure 1.13 – Diagramme de Venn des differents types d'activités [**Figure 1b** ; Mens et Goeminne 2011]

Les participants d'un projet peuvent contribuer sur un projet de plusieurs façons qui sont elles-mêmes détectables avec les méthodes que nous avons décrites en section 1.3.1. Ils peuvent soumettre des changements au code qui sont détectables sur le dépôt de version. Ils peuvent aussi utiliser les différents canaux de communications. Ceci peut être détecté au moyen de l'historique de communication par mail ou via

les forums de discussion. Les outils de bug tracking et de collaboration peuvent aussi être utilisés pour filtrer les communications autour d'une tâche à accomplir.

Il est donc bon de noter qu'une méthode basée sur une seule de ces informations ne permettrait pas d'observer la totalité des développeurs d'un projet.

[Bird et al. 2006] construisent des réseaux sociaux à partir de la mailing list des développeurs Apache. Ils constatent que les développeurs qui communiquent le plus à la mailing list sont aussi ceux qui contribuent le plus au code. Au contraire, une collaboration sans discussion est observée par [Bolici, Howison et Crowston 2009] sur deux projets open source. Ils constatent que, contre intuitivement, pour des tâches concernant de multiples acteurs la plupart sont effectués en l'absence de communication visible entre développeurs. Ils postulent que les développeurs de projets open-source se coordonnent au travers des artefacts qu'ils partagent. Ils mettent en garde quand aux résultats possibles de l'analyse de congruence socio-technique proposée par [Valetto et al. 2007] dans les projets open-source où les membres d'un projet collaborent sans communiquer.

Il est aussi donc bon de considérer les manières de communiquer entre les développeurs qui peut être différente entre les projets.

1.3.3 Détection d'aspects sociaux d'un projet DISS

Les approches abordées jusqu'ici tiennent compte du code source, des communications entre les développeurs (qu'elles soient globales ou en relation avec une tâche spécifique) et les similitudes entre des projets pour établir des liens de collaboration entre les développeurs. Certains travaux tiennent compte de l'historique du projet pour analyser la manière dont cette collaboration évolue au fil du temps. A notre connaissance, aucune de ces méthodes ne tient compte de l'artefact base de données existant dans un DISS pour établir des liens de collaboration entre les développeurs.

Néamoins, des travaux centrés sur l'analyse historique des DISS font état de quelques comportements sociaux que peuvent avoir les développeurs.

[Goeminne, Decan et Mens 2014] analysent la co-évolution entre le code et le schéma de la base de données. Ils groupent les développeurs selon **le type de fichiers qu'ils ont modifiés**.

— les fichiers de code non-liés aux accès à la base de données ;

— les fichiers de code liés aux accès à la base de données ;

— des changement apportés aux deux types de fichiers.

Ils tentent de détecter si les développeurs sont actifs de manière exclusive, soit sur des activités liées à la base de données, soit sur des activités non liées à cette dernière. Il apparait que pour le cas étudié dans leurs recherches, il n'y a pas de séparation entre les développeurs et que tous sont actifs tant sur le code lié que sur

le code non-lié aux accès à la DB. Du point de vue du code, ils analysent l'usage de différentes technologies d'accès à la base de données et son évolution au fil du temps. Ils groupent les développeurs qui utilisent une ou plusieurs technologie(s), mais cette analyse ne tient pas compte de l'aspect collaboratif entre les développeurs ni du schéma de la base de données.

1.4 Vue d'ensemble de l'existant

Nous avons résumé les méthodes de détection de comportements collaboratifs entre développeurs avec le schéma en figure 1.14.

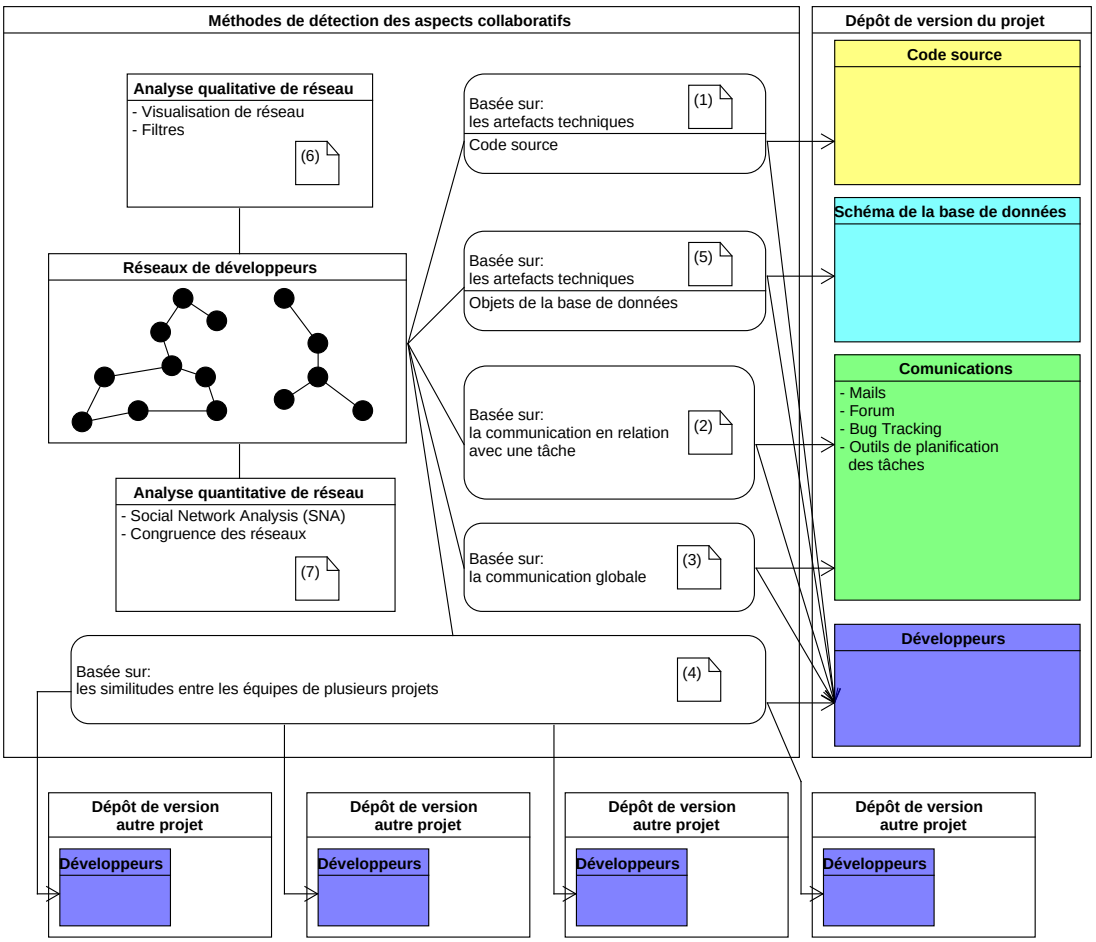


FIGURE 1.14 – Méthodes de construction d'un réseau de développeurs pour un DISS

Dans ce schéma, nous mentionnons les méthodes 1,2 et 3 catégorisées par [WOLF et al. 2009]. Nous évoquons aussi la méthode 4 proposée par [OHIRA et al. 2005]. Concernant le schéma de la base de données, avec ce travail, nous proposons une méthode de construction de réseau de développeurs au moyen des objets de la base de données (i.e. tables ou colonnes). Il s'agit de la méthode 5. En résumé :

[1] les méthodes basées sur les artefacts techniques : code source ;

[2] les méthodes basées sur les communications globales ;

[3] les méthodes basées sur les communications avec les tâches ;

[4] les méthodes basées sur les similitudes entre les équipes de plusieurs projets ;

[5] les méthodes basées sur les artefacts techniques : objets de la base de données.

Les résultats de ces méthodes peuvent être des **réseaux de développeurs** ou des réseaux mixtes. Les réseaux mixtes sont alors constitués de développeurs et d'une autre entité. Cette entité peut être équivalente à celle qui a été utilisée pour faire les liens entre les développeurs. De tels réseaux peuvent être pondérés ; le poids des arrêtes étant le reflet de la force du lien entre les noeuds. Ceci permet de filtrer ou de grouper les noeuds. Les données de ces réseaux peuvent être représentées sous forme d'information via deux types d'analyses :

[6] l'analyse qualitative de réseaux ;

[7] l'analyse quantitative de réseaux.

L'analyse qualitative correspond à la visualisation et éventuellement au filtrage avant visualisation, comme dans la plupart des travaux mentionnés dans la section 1.3.1. L'analyse quantitative, quant à elle, correspond à des méthodes qui fournissent des résultats numériques. Ceux-ci peuvent être transformés en statistiques telles que les méthodes de SNA utilisées par [Martinez-Romo et al. 2008] ou l'analyse de congruence de réseaux utilisée par [Valetto et al. 2007].

1.5 Conclusion

Ce chapitre est important pour comprendre les méthodes de détection des comportements collaboratifs au moyen des méthodes de MSR. Via une revue de littérature, nous avons cherché à compiler les méthodes de construction et d'analyse des réseaux de développeurs et d'en saisir les apports et les limites. D'une part, [Mens et Goeminne 2011] ont déjà étudié les aspects sociaux, mais se sont focalisé sur le code accédant aux bases de données et n'ont pas utilisé de méthodes de SNA. D'autre part, il existe à l'heure actuelle d'intéressantes techniques concernant les aspects collaboratifs sur le développement du code d'un SI. Ces techniques utilisent principalement les informations que procurent les réseaux sociaux des développeurs [Wolf et al. 2009][Ohira et al. 2005][Martinez-Romo et al. 2008]. Cependant, à notre connaissance, les aspects collaboratifs du développement du schéma de la base de données n'ont pas encore été investigués au moyen de méthodes d'analyse de réseaux sociaux. Notre contribution personnelle visera donc à apporter une nouvelle pierre à l'édifice des connaissances pratiques en élaborant un prototype, qui a pour objectif de construire des réseaux de développeurs contribuant au schéma de la base de données et de les analyser.

Chapitre 2

Comportements collaboratifs

2.1 Introduction

Comme exposé dans le chapitre précédent, il existe des outils permettant d'analyser les processus collaboratifs lors du développement du code d'un SI. De tels outils apportent une réelle plus-value lors de la gestion des efforts d'une équipe. En effet, grâce à ces outils il est possible d'obtenir des informations à propos du degré de coordination d'une équipe [Martinez-Romo et al. 2008]. Il est également possible de comprendre le succès ou l'échec du travail effectué par l'équipe pour accomplir une tâche, et même, de prédire le succès ou l'échec du travail en cours en se basant sur les comportements de collaboration [Wolf et al. 2009].

L'objectif de ce travail est de transposer l'utilisation de ces outils aux Data-Intensive Software Systems (DISS). Effectivement, l'analyse de comportement collaboratif lors du dévelopement du schéma de la base de données reste actuellement peu exploré. Or, dans le cas d'un DISS, cette analyse pourrait permettre d'apporter les mêmes bénéfices lors des efforts de développement du schéma de la base de données et du code y accédant.

2.2 Questions de recherche

Nous estimons que l'outil Tesseract [Sarma et al. 2009] est l'exemple qui applique, de manière intéressante, le plus grand nombre de méthodes de Social Network Analysis (SNA). En effet, comme les travaux de [Bird et al. 2006], [Bird et al. 2008], il construit un réseau de communication entre les développeurs en se basant sur les communications mail. Il construit un réseau de fichiers selon leurs relations entre-eux en utilisant une méthode d'analyse statique de **call-graph analysis**[De Souza, Froehlich et Dourish 2005]. Cet outil établit également des liens entre le réseau de développeurs et le réseau de fichiers, comme proposé par [Valetto et al. 2007]. Il permet d'observer une période de développement bien précise et de corréler tous les réseaux obtenus avec d'autres informations, telles que le nombre de bugs et leur niveau de gravité, ce qui est assez semblable à l'approche de [Wolf et al. 2009].

Nous pensons qu'il serait intéressant, dans le domaine des DISS, d'avoir un outil aussi complet que Tesseract. Cet outil posséderait donc les fonctionnalités suivantes :

1. établir des liens entre les développeurs au travers des objets du schéma qu'ils ont modifiés ;
2. établir des liens entre les développeurs au travers des accès à la base de données qu'ils ont modifiés ;
3. filtrer temporellement les liens entre les développeurs, comme par exemple exclure les relations entre eux s'ils ne sont pas contemporains ;
4. établir des liens entre les réseaux obtenus au travers du schéma et ceux obtenus au travers des accès à la base de données. L'objectif est alors d'aider à la co-évolution schéma-code ;
5. établir des liens entre les développeurs au moyen des communications qu'ils échangent entre eux. L'objectif est alors de corréler les patterns de communication aux patterns de modification des artefacts techniques.

Répondre précisement à l'ensemble de ces questions dépasse la portée d'un mémoire. Il serait néanmoins intéressant d'investiguer davantage ces aspects dans des travaux futurs.

Nous avons donc priorisé nos propres investigations en tenant compte des données suivantes :

— [Mens et Goeminne 2011] se sont déjà penchés sur la détection des accès aux bases de données ;

— [Bird et al. 2006] et [Wolf et al. 2009] ont déjà porté leur attention sur les communications et leurs correlations avec les artefacts techniques ;

— [Valetto et al. 2007] ont déjà formalisé la comparaison de réseaux, ce qui a été validé plus tard par les travaux de [Sarma et al. 2009].

L'originalité de notre travail réside donc dans la construction et l'étude de réseaux sociaux obtenus grâce aux modifications apportées au schéma de la base de données (questions 1 et 3 du tableau 2.1). Nous répondrons donc plus précisement à cet aspect, et nous aborderons à plus haut niveau les autres questions de recherche (questions 2,4,5).

TABLE 2.1 – Démarche de questionnement

Bénéfices mis en avant lors des travaux précédents	Intéret pour les DISS	Question de recherche :
Construire un réseau social à partir des artefacts techniques peut apporter des informations interéssantes sur le degré de coordination d'une équipe de développeurs [MARTINEZ-ROMO et al. 2008].	Evaluer le degré de coordination lors du développement du schéma de la base de données.	1) Peut-on détecter les développeurs qui ont effectué des changements au schéma de la base de données sur des objets communs (tables, colonnes) ?
	Evaluer le degré de coordination lors du développement du code accédant à la base de données.	2) Peut-on détecter les développeurs ayant effectué des changements dans le code du programme sur des appels communs à la base de données ?
	Différencier un véritable lien de collaboration de la poursuite du travail initié par une personne ayant quitté le projet.	3) Peut-on déterminer quels développeurs ont été contemporains ?
Un degré d'alignement entre deux réseaux peut donner un moyen rapide d'évaluer à quel point la sous-division des responsabilités dans un projet est respectée [VALETTO et al. 2007]	Evaluer si le responsable d'une modification dans le schéma propage bien le changement jusqu'au code accédant à la partie du schéma qui a changé.	4) Les comportements collaboratifs repris aux questions 1 et 2 sont-ils semblables ?
Identifier les problèmes de collaboration et de communication peut permettre de prédire le succès ou l'échec du travail en cours [WOLF et al. 2009]	Permettre d'identifier des problèmes de collaboration et de communication quand une tâche implique un changement qui impacte le schéma de la base de données et le code y accédant.	5) Peut-on détecter des communications formelles entre développeurs qui prouvent que plusieurs développeurs ont collaboré lors du développement du Système d'information (SI) ?

2.3 Méthodologie

Notre approche générale est représentée en figure 2.1. L'outil **DA**tabase sc**H**ema evo**L**ut**I**on **A**nalysis (DAHLIA)[Meurice et Cleve 2014], dont le fonctionnement sera décrit en section 2.3.1, fournit des données historiques provenant du dépôt de version d'un DISS. Nous utiliserons ces données pour construire des réseaux de développeurs qui peuvent servir d'information quand aux aspects collaboratifs entre ceux-ci.

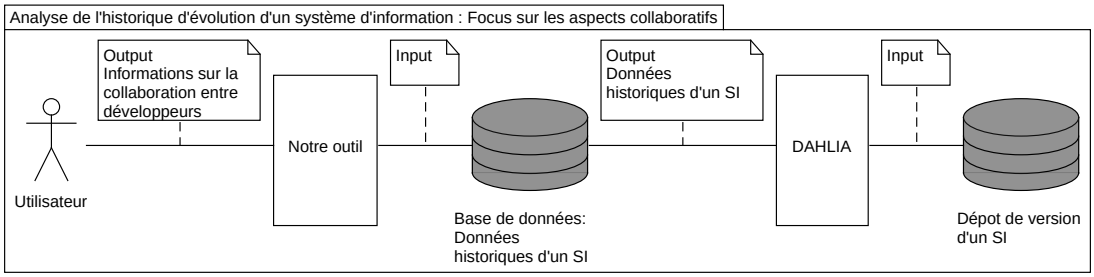


Figure 2.1 – Approche générale

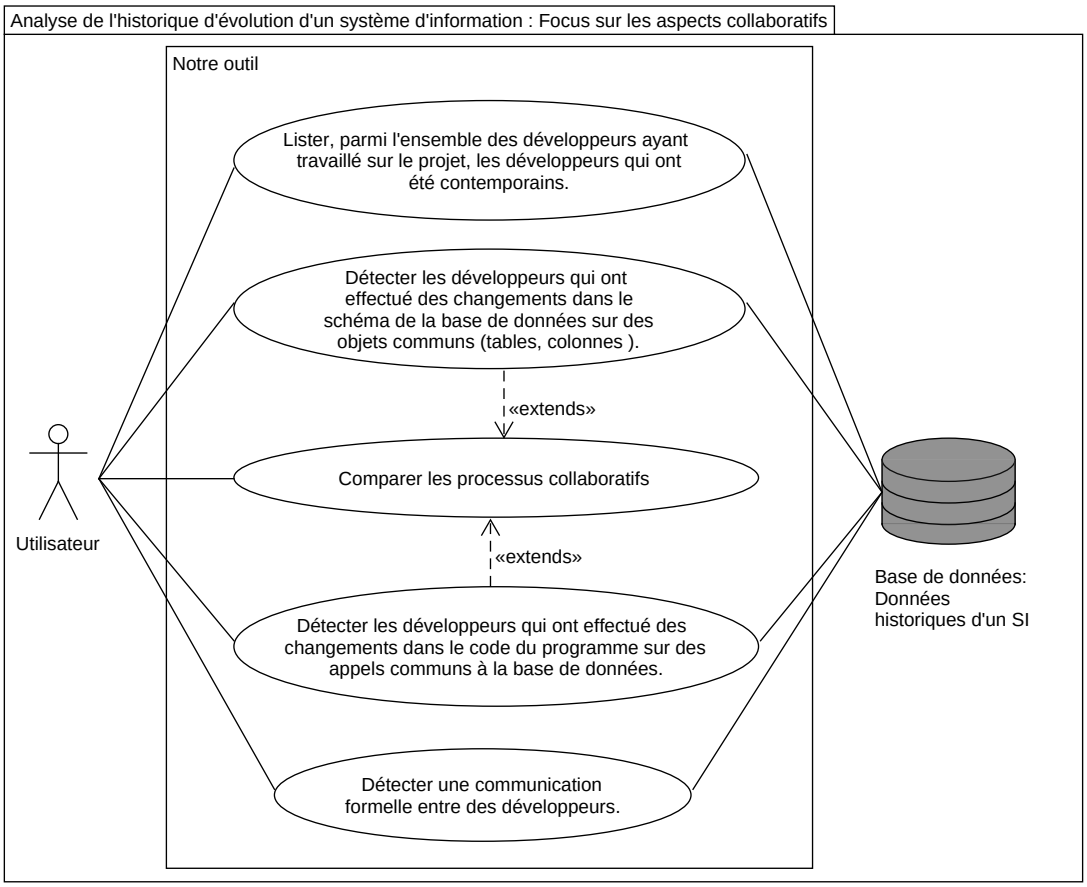


Figure 2.2 – UC diagramme : Approche générale

Sur la figure 2.2 notre outil est représenté avec un diagramme de Use Case. L'utilisateur obtiendra de notre outil, des réseaux qui pourront être représentés en deux formats. Premièrement, il pourra prendre la forme d'un tableau de relations enre-

gistré au format comma-separated values (CSV). Deuxièmement, il pourra prendre la forme d'un graphe enregistré au format DOT qui sont des fichiers pouvant être manipulés et visualisés à l'aide d'un logiciel comme Graphviz [1].

A titre d'évaluation de notre approche, nous avons à notre disposition les données historiques de deux DISS réels (i.e. Broadleaf [2] et OpenMRS [3]) [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*]

Broadleaf

Broadleaf est un logiciel open-source qui offre une plateforme de eCommerce. Le modèle de la base de données est consultable sur le site du projet [4]. Il comporte 178 tables dans la dernière version de l'historique.

OpenMRS

OpenMRS est un logiciel open-source visant à fournir un dossier médical électronique (Electronic Medical Record (EMR)) pour améliorer les soins de santé dans des environements à faibles ressources financières. Le modèle de la base de données est consultable sur le wiki du projet [5]. Il comporte 88 tables dans la dernière version de l'historique à notre disposition.

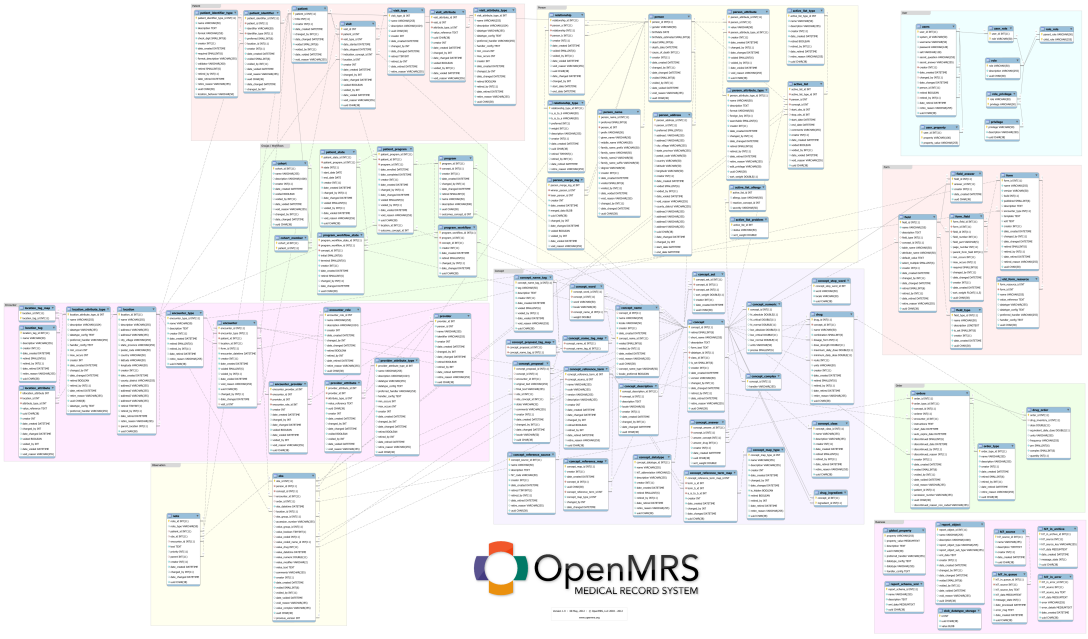


FIGURE 2.3 – Modèle de la base de données de OpenMRS [*OpenMRS Data Model 1.9.0*]

1. `www.graphviz.org`
2. `https://github.com/BroadleafCommerce/BroadleafCommerce/`
3. `https://github.com/openmrs/openmrs-core/`
4. https ://www.broadleafcommerce.com/docs/core/current/database-model/database-model-index
5. https ://wiki.openmrs.org/display/docs/Data+Model

2.3.1 DAHLIA, un outil de visualisation de l'évolution du schéma de bases de données.

L'outil **DA**tabase sc**H**ema evo**L**ut**I**on **A**nalysis (DAHLIA) [Meurice et Cleve 2014] est un outil Mining Software Repositories (MSR) de visualisation de l'évolution du schéma de bases de données. Cet outil fonctionne en quatres étapes :

1. *SQL code extraction* : Extraire tous les fichiers Structured Query Language (SQL) correspondants à chaque version en exploitant le dépôt de version du système.(e.g., GIT/SVN).
2. *Schema Extraction* : Extraire le schéma physique correspondant à chaque fichier SQL.
3. *Historical schema extraction* : Construire le schéma historique en comparant les schémas physiques successifs.
4. *Visualization & exploitation* : Le schéma historique peut être visualisé et interrogé pour obtenir des informations utiles sur l'évolution du schéma de la base de données.

2.3.2 Schéma conceptuel des artefacts d'un DISS

Le schéma historique construit par l'outil DAHLIA est décrit par le schéma conceptuel [Meurice et al. 2016] représenté en figure 2.4. Il est composé des différents artefacts qui peuvent être analysés historiquement dans un DISS. Nous le détaillerons dans la section 2.3.3

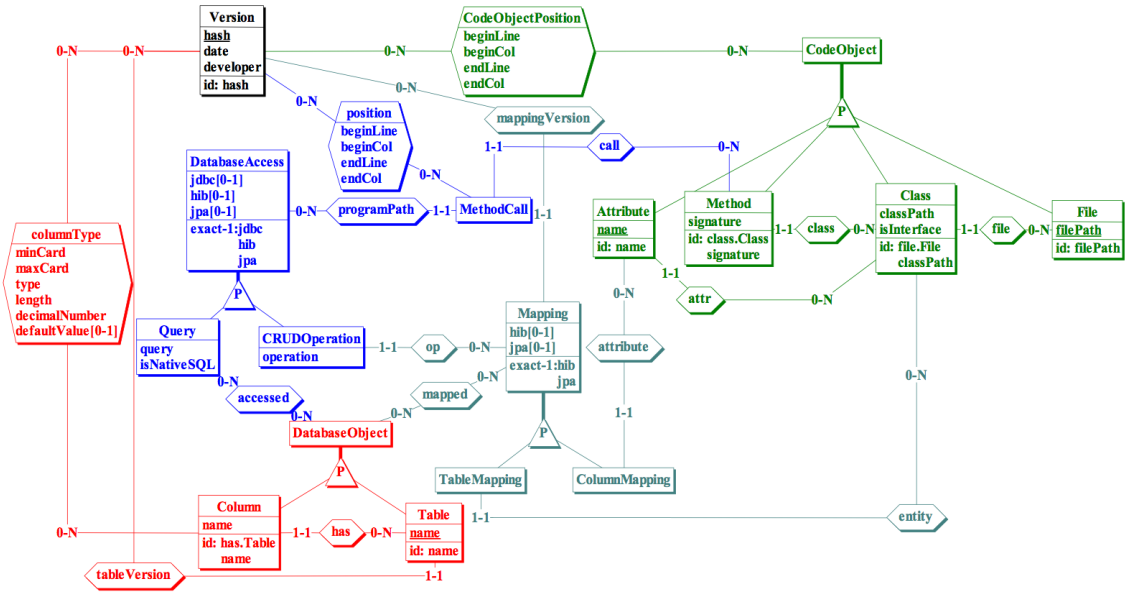


Figure 2.4 – Schéma conceptuel [Meurice et al. 2016]

2.3.3 Description du schéma conceptuel

Pour les projets open-source OpenMRS [6] et Broadleaf [7] nous avons à notre disposition les données historiques fournies par l'outil DAHLIA. Les données historiques nous ont été fournies au format SQL [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*] et respectent le shéma conceptuel en figure 2.4.

L'élément central de ce schéma est le concept de **Version** qui reprend l'identifiant du **developer** qui a soumis celle-ci.

Le concept de **schéma de la base de données** d'un projet informatique est illustré en figure 2.5. **DatabaseObject** représente un objet du schéma de la base de données. Il peut être une **Column** ou une **Table**. Chaque objet peut exister dans plusieurs **Version** du schéma **columnType** et **tableVersion**. Une **table** contient plusieurs **Column**. Une **Column** peut avoir un typage différent en fonction des versions du schéma, l'historique de son typage etant contenu dans **ColumnType**.

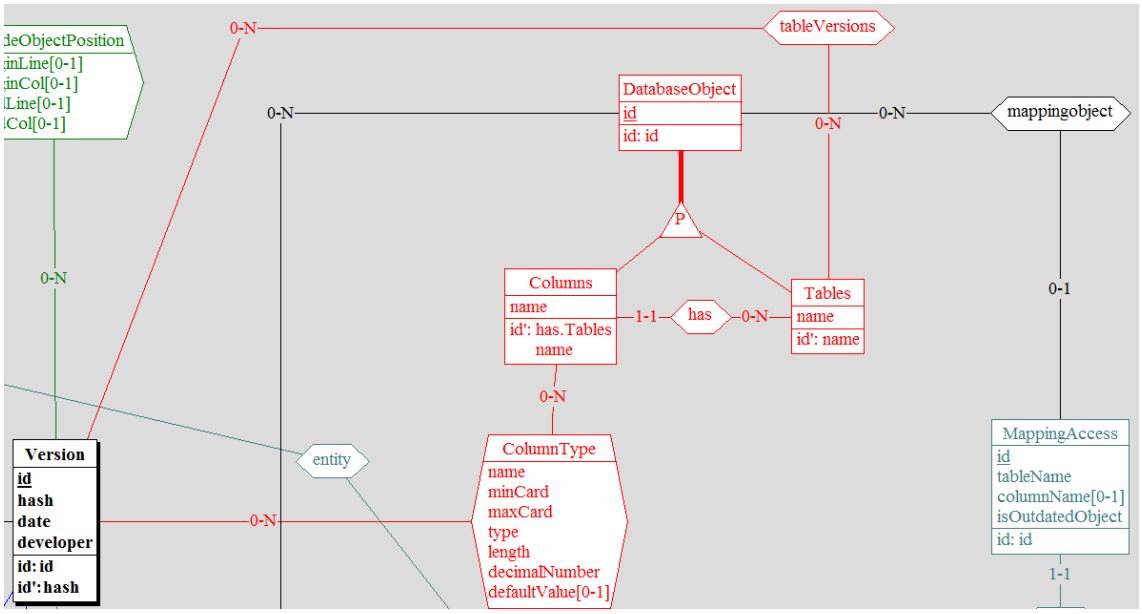


FIGURE 2.5 – Schéma logique, partie schéma de la base de données [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*]

Grâçe à ces données, il est possible de faire le lien entre des développeurs ayant modifié le ou les même(s) objet(s) de la base de données tel qu'illusté en figure 2.6.

6. `https://github.com/openmrs/openmrs-core/`
7. `https://github.com/BroadleafCommerce/BroadleafCommerce/`

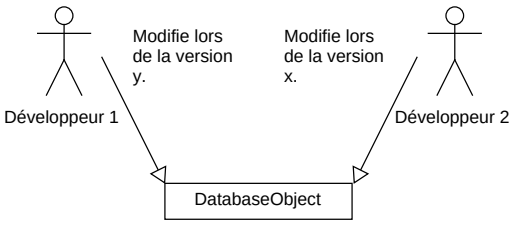


FIGURE 2.6 – Lien entre des développeurs au travers des modifications éffectuées à un objet de la base de données.

Le concept du **code JAVA** d'un projet informatique est illustré en figure 2.7. **CodeObject** représente un objet du code du système. Le projet est divisé en **SubProject** qui contient plusieurs **File** JAVA qui eux-mêmes peuvent contenir plusieurs **Class**. Une Class peut contenir plusieurs **Method** JAVA. Chaque objet du code peut être présent dans plusieurs **Version**. Leur position **CodeObjectPosition** est exprimée par un couple de coordonnées (beginLine,beginCol),(endLine,endCol) délimitant leur position dans le fichier JAVA.

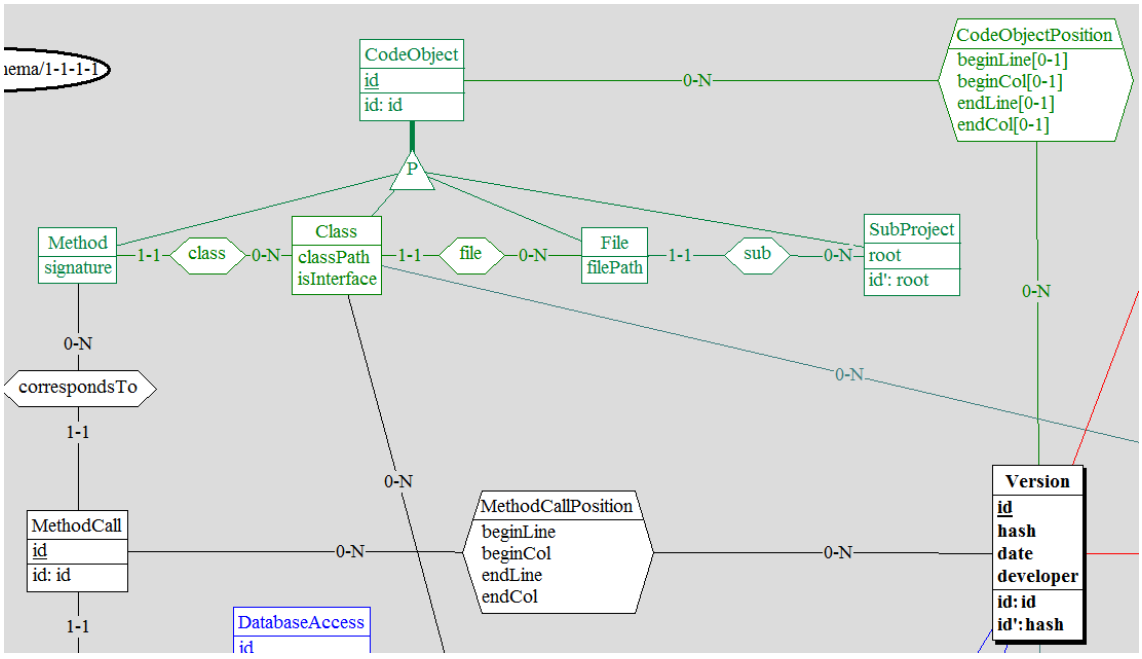


FIGURE 2.7 – Schéma logique, partie code JAVA [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*]

Grâçe à ces données, il est possible de faire le lien entre des développeurs ayant modifiés un ou plusieurs objet(s) identique(s) du code JAVA tel qu'illustré en figure 2.8.

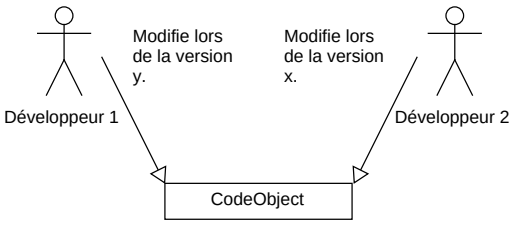


FIGURE 2.8 – Lien entre des développeurs au travers des modifications éffectuées à un objet du code JAVA.

Le concept des **accès à la base de données** est représenté en figure 2.9. Les accès sont soit du type **Query** lors d'un accès au moyen de jdbc, soit du type **CRUDOperation** lors d'un accès utilisant un mapping Object-Relational-Mapping (ORM) au moyen de Hibernate ou JPA. Chaque Query ou CRUDOperation peut changer selon les versions, ce qui est stocké au moyen des entités **VersionQuery** et **VersionCRUD**. De tels changements peuvent être attribués à une **Version** et donc à un **developer**.

Grâce à ces données, il est possible de faire le lien entre le code et la base de données :

— pour un accès jdbc, au moyen de **QueryAccess** qui fait le lien entre VersionQuery et DatabaseObject ;

— pour un accès hibernate ou jpa au moyen de **CRUDOperation** et DatabaseObject au travers d'un mapping ORM.

Il est donc possible d'établir un lien entre des développeurs si ils ont modifié le ou les mêmes accès à la base de données. Il est également possible d'identifier les technologies utilisées par les développeurs et d'utiliser cette information pour organiser les résultats.

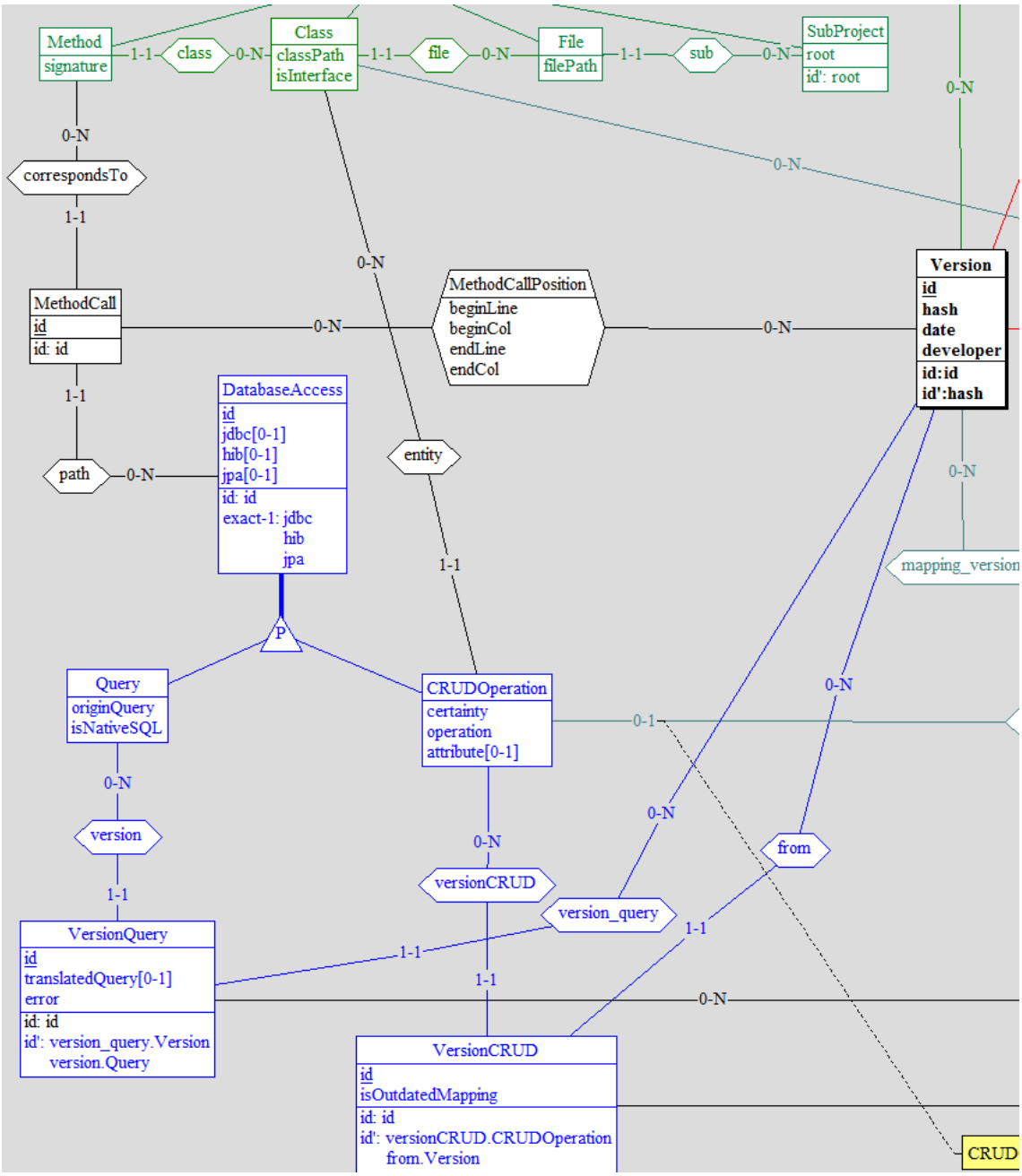


FIGURE 2.9 – Schéma logique, partie accès à la base de données [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*]

Le concept de **mapping ORM (JPA ou Hibernate)** est illustré en figure 2.10. Il existe deux types de mapping :

— Un mapping entre une classe JAVA et une table.

— Un mapping entre un attribut JAVA et une colonne.

Le comportement est défini par **MappingAccess**. Il peut être différent selon les versions, ce qui est représenté par **MappingVersion**.

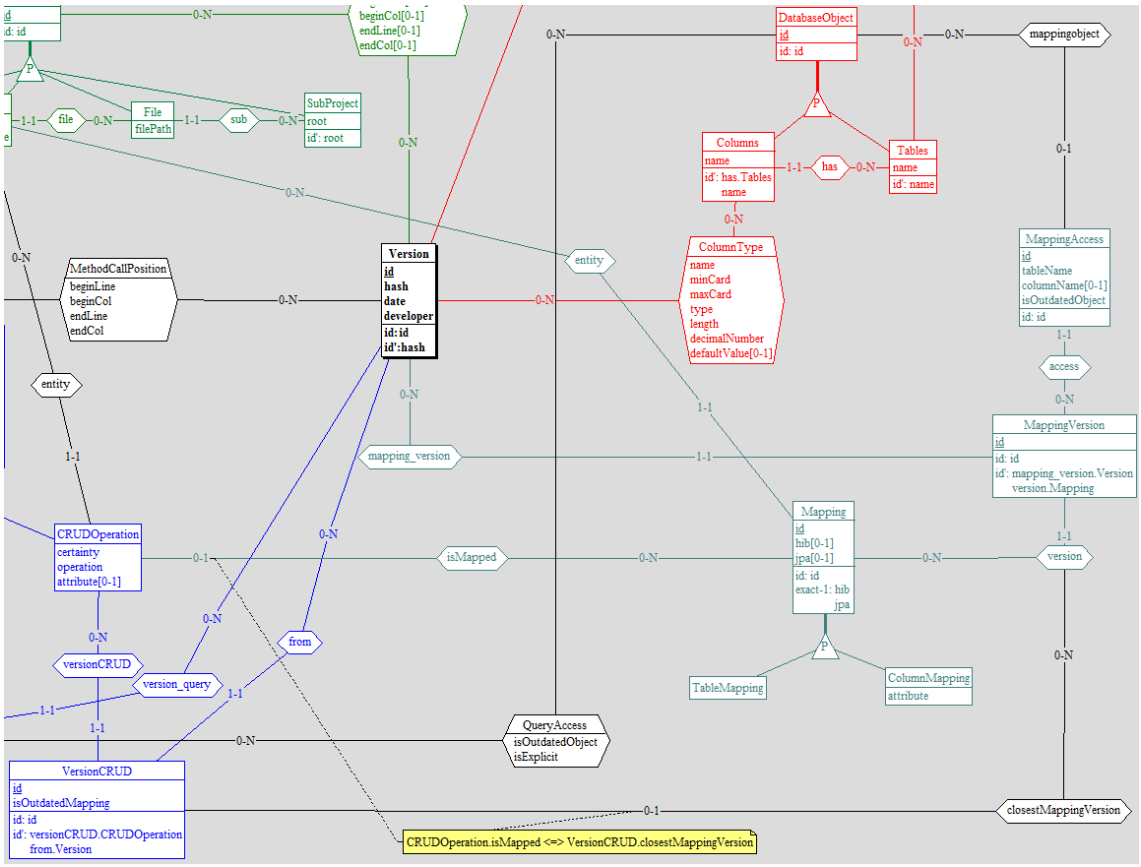


FIGURE 2.10 – Schéma logique, partie mapping ORM [*Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.*]

2.4 Conclusion

Pour résumer ce chapitre, nous mettons en relation dans la table 2.2 les questions que nous nous sommes posées et les éléments de données à notre disposition pour tenter d'y répondre.

Question de recherche	Données disponibles ?
1) Peut-on détecter les développeurs qui ont effectué des changements au schéma de la base de données sur des objets communs (tables, colonnes) ?	Oui, au moyen des données de la partie du schéma conceptuel qui décrit le schéma de la base de données en figure 2.5
2) Peut-on détecter les développeurs ayant effectué des changements dans le code du programme sur des appels communs à la base de données ?	Oui, au moyen des données des parties du schéma conceptuel qui décrivent le code, les accès à la base de données, les mapping ORM.
3) Peut-on déterminer quels développeurs ont été contemporains ?	Oui, au moyen des données contenues dans la table Version : Date et developer
4) Les comportements collaboratifs repris aux questions 1 et 2 sont-ils semblables ?	Au moyen des réponses aux questions 1 et 2, et en faisant le lien entre le code et le schéma avec les informations des accès à la base de données et les mapping ORM.
5) Peut-on détecter des communications formelles entre développeurs qui prouvent que plusieurs développeurs ont collaboré lors du développement du SI ?	Non, il n'y a pas de données relatives aux communications formelles dans le schéma conceptuel.

TABLE 2.2 – Questions de recherche et données à notre disposition pour y répondre

Les questions 1 et 3 nous interéssent plus particulièrement car ce domaine a été peu exploré jusqu'a présent. Nous développerons une méthode et un outil que nous évaluerons avec les données historiques de deux DISS mises à notre disposition.

La question 2 revient à construire des réseaux de développeurs en se basant sur le code source, plus particulièrement sur les appels à la base de données.

Pour aborder la question 4 il est nécessaire de tenir compte des données produites par les réponses aux questions 1 et 2. Répondre à cette question nécessite également d'utiliser une méthode telle que décrite dans [VALETTO et al. 2007] pour évaluer la congruence entre les deux réseaux. Cette méthode permet d'évaluer le dégré d'alignement des relations entre développeurs au niveau du code avec les relations entre développeurs au niveau du schéma. Nous donnerons des pistes de réflexion sur les moyens à mettre en oeuvre pour répondre à cette question.

La question 5 revient à construire des réseaux de développeurs au moyen des

communications globales concernant le projet. [Bird et al. 2008] ont déjà éffectué ce type de travaux et, plus récement, [Wolf et al. 2009] ont affiné ce type de travaux en utilisant les communications en relation avec une tâche. Il n'existe pas de données relatives aux communications formelles dans le schéma conceptuel de [Meurice et al. 2016]. Nous proposerons donc d'enrichir ce schéma conceptuel pour acceuillir ce type de données dans le futur.

Chapitre 3

Contribution personnelle

3.1 Introduction

Intuition : Dans les données reprises par le schéma conceptuel de [Meurice et al. 2016] nous ne sommes pas en possession des données historiques des communications formelles entre les développeurs telles que email, bug tracking, etc. Le schéma conceptuel comporte l'historique des versions du schéma, des accès à la base de données, les mapping ORM et le code java. Néanmoins, nous pensons qu'il est possible de détecter un comportement de collaboration à partir des modifications apportées au shéma et au code au fur et à mesure des versions. En partant du principe que si plusieurs développeurs travaillent sur des parties communes du schéma ou du code, il serait possible d'inférer qu'ils ont collaboré sur une partie commune et probablement communiqué à son sujet.

3.2 Limites des données à disposition et hypothèses simplificatrices

Notre outil posera les hypothèses simplificatrices suivantes ;

— les données historiques utilisées lors de notre évaluation ne comportent pas un historique de versions jointives. Les versions absentes peuvent induire un biais car les changements apportés lors des versions absentes peuvent être attribués à un autre développeur que celui qui les a implémentés.

— Nous considérons que les développeurs utilisent toujours un seul identifiant lorsqu'ils travaillent sur le projet.

A la fin du chapitre, nous explorerons des pistes de solutions pour lever ces hypothèses.

3.3 Question de recherche 1 : Peut-on détecter les développeurs qui ont effectué des changements au schéma de la base de données sur des objets communs ?

Notre objectif est d'établir des liens entre des développeurs qui permettraient de supposer qu'ils ont collaboré car ils ont travaillé sur une partie commune du schéma de la base de données. Pour atteindre cet objectif, nous allons détecter l'existence ou l'absence de relations entre deux développeurs "à apporter des changements à un même objet du schéma (table ou colonne)".

Nous voulons visualiser les résultats en un tableau de relations représenté comme une table à deux dimensions égales comme illustré en table 3.1a ou en graphe comme illustré en figure 3.1b. Les lignes et les colonnes correspondent aux développeurs travaillant sur le projet. Deux groupes de développeurs seront distingués selon la valeur de la cellule commune aux deux developpeurs :

— si la valeur est à 1, les développeurs ont apporté des changements à un même objet (table, colonne) du schéma de la base de données ;

— si la valeur est à 0, les développeurs n'ont pas apporté de changements à un même objet (table, colonne) du schéma de la base de données.

Dans le cas du graphe, s'il y a une arrête entre deux développeurs, alors ils ont apporté des changements à un même objet (table, colonne).

	D0	D1	D2	...	Dn
D0	0	0	1	...	0
D1	0	0	1	...	0
D2	1	1	0	...	0
...	...	...	...	0	...
Dn	0	0	0	...	0

(a) Visualisation en tableau de relations.

(b) Visualisation en graphe

FIGURE 3.1 – Visualisation de relations "à apporter des changements à un même objet du schéma (table ou colonne)"

3.3.1 Méthodologie

Pour atteindre notre objectif nous allons procéder aux analyses suivantes :

1. En section 3.3.2 nous aborderons l'analyse des changements apportés aux tables du schéma :

— recherche de modifications des tables, attribution de ces changements à un développeur et visualisation des résultats ;

— etablir des liens entre les développeurs à l'aide des modifications à une table commune et visualisation des résultats.

2. En section 3.3.5 nous aborderons l'analyse des changements apportés aux colonnes :

— recherche de modifications des colonnes appartenant aux tables, attribution de ces changements à un développeur et visualisation des résultats ;

— établir des liens entre les développeurs à l'aide des modifications de colonnes dans une table commune et visualisation des résultats.

3.3.2 Les changements apportés aux tables du schéma

Principe de détection des changements

Tout d'abord, nous allons définir quels changements peuvent survenir pour une table du schéma d'une base de données. On considère comme changement n'importe laquelle de ces opérations :

— l'ajout, au moyen de CREATE TABLE ;

— la suppression, au moyen de DROP TABLE ;

— le renommage, au moyen de ALTER TABLE x RENAME TO y ;

— la fusion de plusieurs tables en une ;

— la scission d'une table en plusieurs.

Par exemple, la répartition des ajouts et des suppressions entre la version 0 et 1 du projet Broadleaf est la suivante :

— 22 tables ont été supprimées. Il s'agit de la différence entre l'ensemble des tables de la version 0 et de la version 1 ($Version0 \setminus Version1$), en rouge sur la figure 3.2.

— 32 tables ont été ajoutées. Il s'agit de la différence entre l'ensemble des tables de la version 1 et de la version 0 ($Version1 \setminus Version0$), en vert sur la figure 3.2.

— 2 tables ont été conservées. Il s'agit de l'intersection entre l'ensemble des tables de la version 0 et de la version 1 ($Version0 \cap Version1$), en bleu sur la figure 3.2.

Les opérations ($Version1 \setminus Version0$) et ($Version0 \setminus Version1$) entre ces deux versions ne sont pas seulement dues aux ajouts et aux suppressions de tables. Le

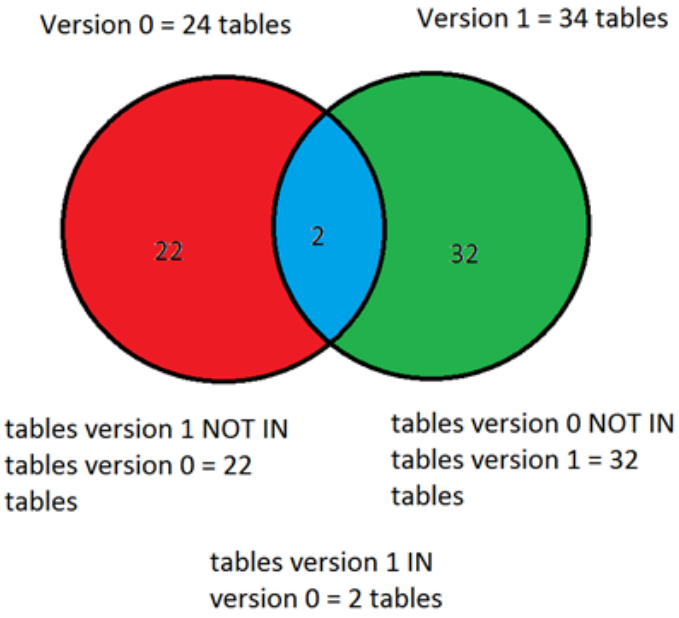


FIGURE 3.2 – Diagrame de Venn des changements entre les version 0 et 1 de Broadleaf

renommage, la fusion et la scission de tables sont repris dans ces ensembles comme une combinaison d'ajouts et de suppressions.

La détection du renommage, de la fusion et de la scission des tables n'est pas une tâche aisée. Elle n'est cependant pas nécessaire car nous voulons seulement détecter les changements apportés aux tables, peu importe qu'ils soient dus à un ajout, une suppression, un renommage ou une scission.

Tous les types de changements font partie de la différence symétrique des ensembles des tables de version 0 et version 1 ($Version1 \triangle Version0$).

Détection des changements dans l'historique de version

Notre but est de construire une table de relations entre l'ensemble des tables ayant existé pendant la durée du projet et l'ensemble des développeurs ayant participé pendant le projet. Le format du tableau, obtenu par notre outil, pour X développeurs et Y tables est représenté par la table 3.1.

TABLE 3.1 – Tableau de relations "La table X a été modifiée par le développeur Y"

	D0	D1	D2	...	Dx
T0	0	1	1	...	0
T1	1	0	1	...	0
T2	0	0	0	...	0
...	...	...	...	...	...
Ty	2	0	0	...	0

L'utilité est de chiffrer le nombre de changements qu'un développeur apporte à

une table en particulier tout au long des versions. Quand plusieurs développeurs apportent un changement à une même table, nous utilisons ceci comme une indication de collaboration entre ces développeurs.

Pour la table 3.1 on constate que :

— Les développeurs *D1* et *D2* ont chacun apporté des modifications à la table *T0*.

— Les développeurs *D0* et *D2* ont chacun apporté des modifications à la table *T1*.

Avec l'information contenue dans la table 3.1, nous pouvons établir des liens entre les développeurs qui ont apporté des changements tels que ajout, suppression, renommage ou scission de mêmes tables dans le schéma de la base de données. Le format du tableau de relations est représenté par la table 3.2.

TABLE 3.2 – Tableau de relations "Développeur X a modifié une même table du schéma que le développeur Y"

	D0	D1	D2	...	Dx
D0	0	0	1	...	0
D1	0	0	1	...	0
D2	1	1	0	...	0
...	...	...	...	0	...
Dy	0	0	0	...	0

— La valeur 1 est attribuée aux cellules communes entre deux développeurs apportant des modifications à une même table. Dans le cas du tableau 3.1 il s'agira des cellules communes aux développeurs :

  — *D0* et *D2*.

  — *D1* et *D2*.

— Les cellules communes entre deux développeurs n'ayant pas apporté de changements à une ou plusieurs table(s) restent à la valeur 0.

— La cellule commune au même développeur est toujours à 0.

Différence entre l'ensemble des tables de la version 1 et de la version 0 ($Version1 \setminus Version0$)

Un "ajout de table" entre la version 0 et la version 1 est détecté grâce à la requête mySQL suivante :

```
# List of the tables that are present in the current version (1)
# and not in the previous version (0)
SELECT tableversions.table_id, version.id
```

```
FROM version, tableVersions, tables
WHERE version.id = tableVersions.version_id
AND tableVersions.table_id = tables.id
AND version.id= 0
AND tableVersions.table_id
IN ( SELECT tableversions.table_id
FROM version, tableVersions, tables
WHERE version.id = tableVersions.version_id
AND tableVersions.table_id = tables.id
AND version.id= 1);
```

Différence entre l'ensemble des tables de la version 0 et de la version 1 ($Version0 \setminus Version1$)

Une "suppression de table" entre la version 0 et la version 1 est détectée grâce à la requête mySQL suivante :

```
# List of the tables that are present in previous version (0)
# and not in current version (1)
SELECT tableversions.table_id, version.id
FROM version, tableVersions, tables
WHERE version.id = tableVersions.version_id
AND tableVersions.table_id = tables.id
AND version.id= 1
AND tableVersions.table_id
IN ( SELECT tableversions.table_id
FROM version, tableVersions, tables
WHERE version.id = tableVersions.version_id
AND tableVersions.table_id = tables.id
AND version.id= 0);
```

Différence symétrique des ensembles des tables de version 0 et version 1 ($Version1 \triangle Version0$)

Pour détecter les changements, nous utilisons l'union de $(Version1 \setminus Version0)$ et de $(Version0 \setminus Version1)$ pour obtenir la différence symétrique des ensembles des tables de version 0 et version 1 ($Version1 \triangle Version0$). Notre outil utilise le *PreparedStatement* suivant :

```
ps = "SELECT tableVersions.table_id " +
        "FROM version, tableVersions, tables " +
```

```
"WHERE version.id = tableVersions.version_id " +
"AND tableVersions.table_id = tables.id " +
"AND version.id= "+current+" " +
"AND tableVersions.table_id " +
"NOT IN ( SELECT tableversions.table_id " +
"FROM version, tableVersions, tables " +
"WHERE version.id = tableVersions.version_id " +
"AND tableVersions.table_id = tables.id " +
"AND version.id= "+previous+") " +
"UNION " +
"SELECT tableVersions.table_id " +
"FROM version, tableVersions, tables " +
"WHERE version.id = tableVersions.version_id " +
"AND tableVersions.table_id = tables.id " +
"AND version.id= "+previous+" " +
"AND tableVersions.table_id " +
"NOT IN ( SELECT tableversions.table_id " +
"FROM version, tableVersions, tables " +
"WHERE version.id = tableVersions.version_id " +
"AND tableVersions.table_id = tables.id " +
"AND version.id= "+current+");";
```

3.3.3 Evaluation de notre méthode sur des cas pratiques

Des tableaux représentant les relations "La table X a été modifiée par le développeur Y" ont été obtenus par notre outil sur de véritables données historiques de systèmes existants. Un tableau de 278 tables (lignes) pour 25 développeurs (colonnes) à été obtenu pour le projet Broadleaf. Un tableau de 100 tables (lignes) pour 43 développeurs (colonnes) a été obtenu pour le projet OpenMRS.

Pour synthétiser l'information, la distribution des valeurs contenues dans les tableaux de relations "La table X a été modifiée par le développeur Y" sont reprises à la table 3.3 pour Broadleaf, à la table 3.4 pour OpenMRS.

TABLE 3.3 – Synthèse du tableau "La table X a été modifiée par le développeur Y" pour Broadleaf

Valeur	# de cellules	% du total	# de changements	% des changements
0	6448	92,78		
1	391	5,63	391	77,89
2	105	1,51	105	20,92
3	4	0,06	4	0,80
4	1	0,01	1	0,20
5	1	0,01	1	0,20
Total	6950	100	502	100

TABLE 3.4 – Synthèse du tableau "La table X a été modifiée par le développeur Y" pour OpenMRS

Valeur	# de cellules	% du total	# de changements	% des changements
0	4194	97,53		
1	100	2,33	100	94,34
2	6	0,14	6	5,66
Total	4300	100	106	100

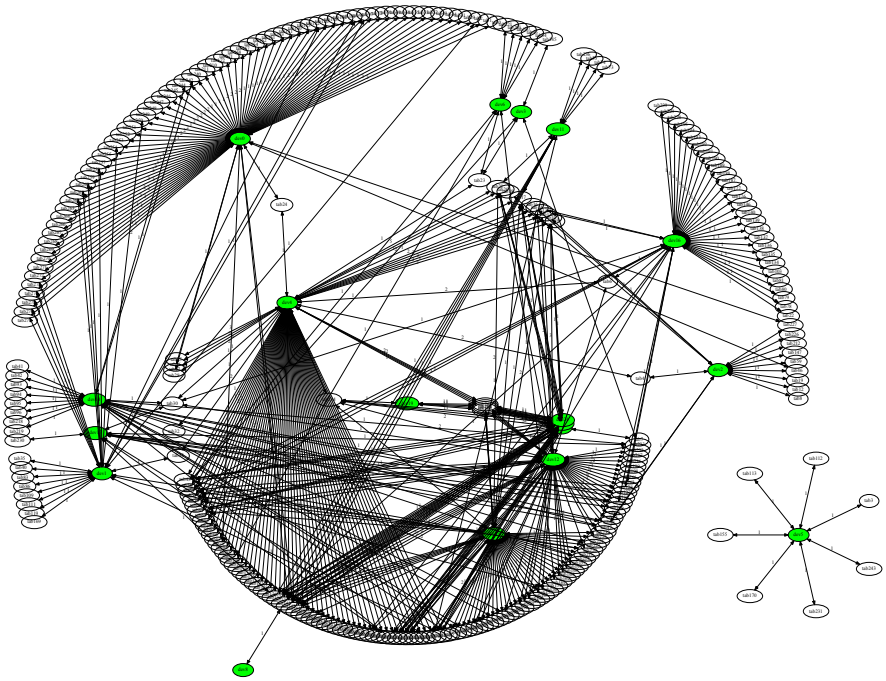
Pour Broadleaf , on constate que :

- il y a eu 502 modifications pour 178 tables encore présentes lors de la dernière version du projet ;
- il y a plus de tables qui ont été modifiées une seule fois par un développeur que de tables présentes lors de la dernière version du projet. Cela veut dire que les développeurs de Broadleaf se permettent souvent de supprimer ou renommer des tables créées par un autre développeur ;
- normalement, une table est touchée une fois lors de sa création, éventuellement une seconde fois en cas de suppression ou de renommage. Si il y a eu plus de 2 changements sur une même table, cela veut dire qu'un même développeur peut avoir supprimé/renommé une table pour ensuite décider de revenir sur sa décision.
  - 4 tables ont été modifiées 3 fois par un même développeur ;
  - 1 table a été modifiée 4 fois par un même développeur ;
  - 1 table a été modifiée 5 fois par un même développeur.

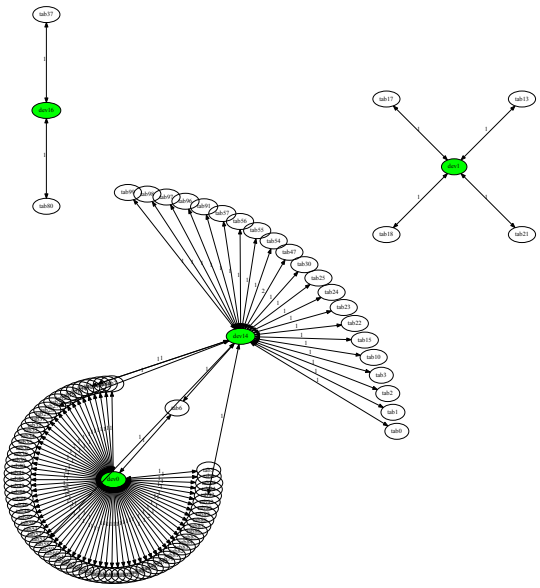
Pour OpenMRS , On constate que :

- Seuls 4 développeurs sur 43 ont modifié une table.
- 2 développeurs sur ces 4 ont modifié une table commune.
- Seules 6 tables ont été touchées deux fois par un même développeur.
- Aucune table n'a été modifiée plus de 2 fois par un même développeur.

Ces tableaux sont de bien trop grande taille que pour être inclus sous forme de texte. Nous les avons représentés sous forme de graphes pour les visualiser. Les tables et les développeurs sont les noeuds du graphe. Les relations "La table X a été modifiée par le développeur Y" sont les arrêtes entre les noeuds. Le poids des arrêtes représente le nombre de modifications. De tels graphes sont représentés en figure 3.3a pour Broadleaf et en figure 3.3b pour OpenMRS.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.3 – Graphe de relations "La table X a été modifiée par le développeur Y"

"Développeur X a modifié une même table du schéma que le développeur Y"

Le tableau de relations tables-développeurs permet d'établir des relations entre développeurs au travers des tables qu'ils ont modifiées en commun. Nous établissons alors un tableau de relations "Développeur X a modifié une même table du schéma que le développeur Y",

Pour Broadleaf , on constate à l'aide de la table de relations 3.5 que 8 développeurs sur 25 (5,7,10,15,18,20,22,23) n'ont pas changé une même table qu'un autre développeur. La table de relations peut être représentée comme un graphe comme illustré en figure 3.4a, ce qui est plus visuel.

Pour OpenMRS , on constate à l'aide de la table de relations 3.6 que seuls les développeurs 0 et 14 ont touché une ou plusieurs tables communes. Tous les autres développeurs n'ont pas "collaboré". Ceci n'est pas étonnant car, globalement, il y a eu peu de suppressions ou de renommages de tables. Le graphe représenté en figure 3.4b est beaucoup plus synthétique que la table de relations.

Globalement on constate que :

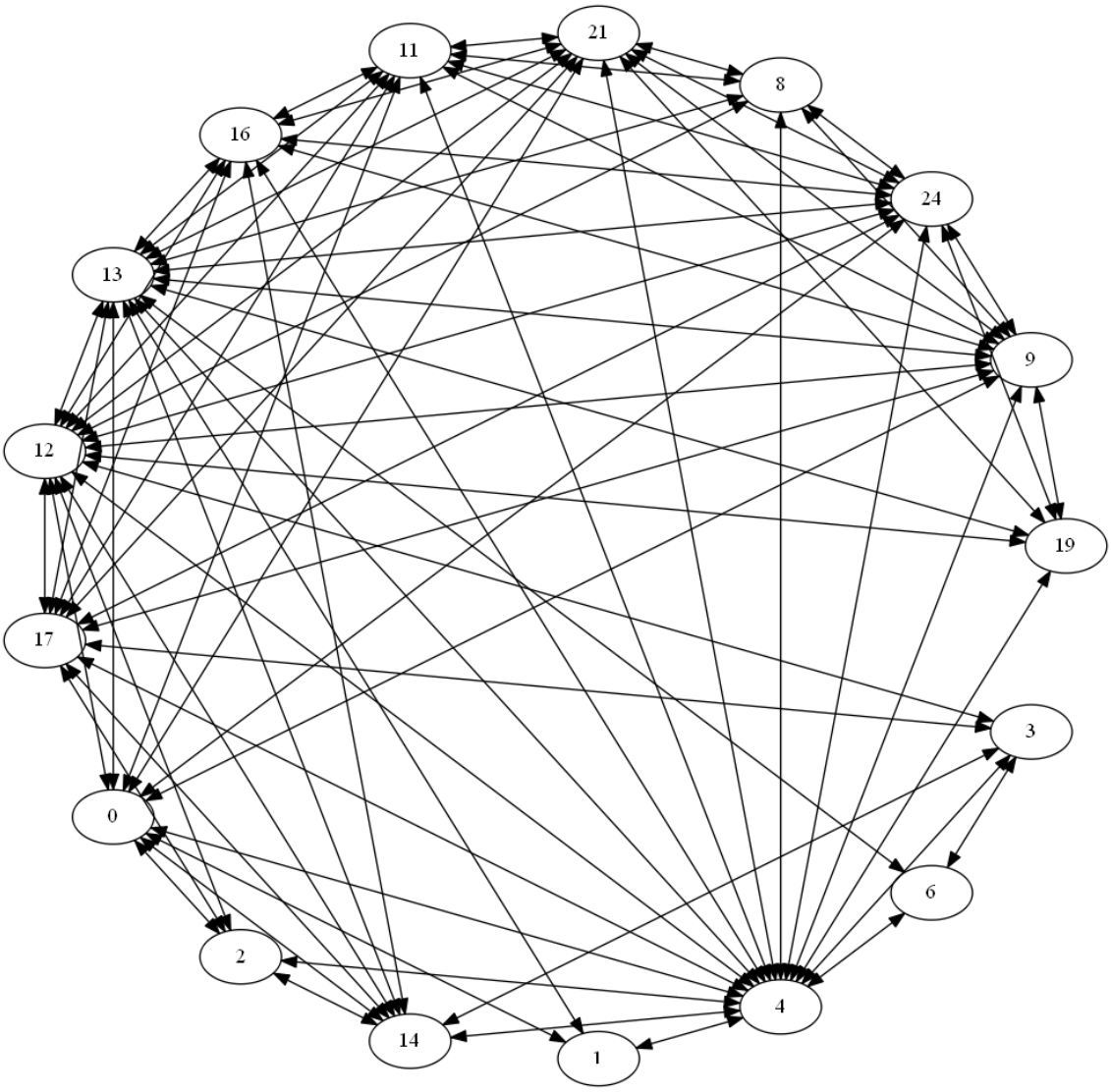
- il y a plus de changements apportés aux tables du schéma pour OpenMRS que pour Broadleaf ;
- contre-intuitivement, il y a peu de lien de collaboration entre les développeurs de OpenMRS pour une forte stabilité du schéma et beaucoup de lien de collaboration entre les développeurs de Broadleaf pour beaucoup de changements du schéma ;
- contre-intuitivement, l'équipe de OpenMRS est plus large (43 developpeurs) que l'équipe de Broadleaf (25 développeurs). On pourrait se dire qu'une équipe plus petite est plus encline à collaborer ;
- on pourrait éventuellement argumenter que le manque de collaboration entre les développeurs de OpenMRS les empêche de supprimer des tables. On peut également penser que la forte collaboration des développeurs de Broaleaf leur permet de supprimer et de renommer des tables qu'il n'ont pas ajoutées eux-mêmes ;
- on constate que, contrairement à Broadleaf, la responsabilité d'ajouter et de supprimer des tables semble être attribuée à un sous-ensemble des développeurs de OpenMRS.

TABLE 3.5 – Table de relations "Développeur X a modifié une même table du schéma que le développeur Y" pour Broadleaf

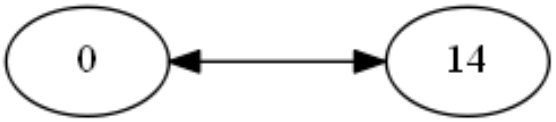
X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	1	1	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0
4	1	1	1	1	0	0	1	0	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	0	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1
9	1	0	0	0	1	0	0	0	1	0	0	1	1	1	0	0	1	1	0	1	0	1	0	0	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1
12	1	0	1	1	1	0	0	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1	0	0	1
13	1	1	0	0	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	0	1	0	0	1
14	1	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0	0	1
17	0	0	1	1	1	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0	1	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	1	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	0	0	0	1
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	1	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	0	0	0

TABLE 3.6 – Table de relations "Développeur X a modifié une même table du schéma que le développeur Y" pour OpenMRS

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



(a) Broadleaf



(b) OpenMRS

FIGURE 3.4 – Graphe des relations "Développeur X a modifié une même table du schéma que le développeur Y"

3.3.4 Amélioration possible

Nous commencons tout d'abord à établir des liens entre développeurs s'ils ont tous les deux apporté au moins un changement à une table commune. Dans le cas où cette méthode engendrerait des *faux positifs*, il est possible d'affiner les résultats en choissisant un nombre de changements apporté à un nombre de tables communes >1. La valeur serait choisie arbitrairement pour diminuer la sensibilité du test et augmenter sa spécificité. Ce nombre arbitraire pourrait être choisi si nous avions une référence à laquelle comparer nos résultats. Les résultats obtenus par la question de recherche 5 pourraient servir de référence.

3.3.5 Les changements apportés aux colonnes des tables du schéma

Principe de détection des changements

Nous cherchons à obtenir un tableau dont le format obtenu est similaire à la table 3.1. Les lignes représentent les tables et les colonnes représentent les développeurs. Ici la relation entre une table X et un développeur Y est "Une ou plusieurs colonne(s) de la table X à été modifiée(s) par le développeur Y".

Pour le construire, nous chiffrons le nombre de changements qu'un développeur apporte à une ou plusieurs colonne(s) d'une table. Un changement peut être l'ajout, la suppression, le renommage ou le changement de typage d'une ou plusieurs colonne(s).

Cette analyse n'est effectuée que pour les tables qui n'ont pas subi de changement lors de l'analyse 3.3.2. Il s'agit de l'intersection entre l'ensemble des tables de la version 0 et de la version 1 ($Version0 \cap Version1$).

Les valeurs typiques des cellules peuvent être plus élevées que l'analyse effectuée en section 3.3.2 car plusieurs colonnes d'une même table peuvent changer au fur et à mesure des versions.

De ce tableau, nous dérivons un graphe de relations entre les développeurs qui ont tous les deux apporté des changements tels que ajout, suppression, renomage ou changement du typage d'une colonne dans une table commune.

Le format du tableau obtenu est similaire à la table 3.2. Dans le tableau de relations entre les développeurs généré par notre outil :

— La valeur 1 sera attribuée aux cellules communes entre deux développeurs qui apportent des modifications à des colonnes appartenant à une même table.

— Les autres cellules restent à la valeur 0.

— La cellule commune au même développeur est toujours à 0.

Détection des changements dans l'historique de version

Comme chaque version contient un cliché du schéma complet de la base de données pour cette version, il faut faire une différence de deux versions consécutives pour obtenir les changements apportés par une version.

Pour générer le tableau de relations "Une ou plusieurs colonne(s) de la table X à été modifiée(s) par le développeur Y", notre outil utilise le *PreparedStatement* suivant :

```
ps = "SELECT * FROM ( " +
"SELECT column_id, columntype.name AS columntype_name,columns.name " +
```

```
"AS columns_name,id,minCard,maxCard,type,length,decimalNumber " +
",defaultValue,table_id FROM columntype, columns  " +
"WHERE columnType.column_id = columns.id  " +
"AND version_id="+previous+" " +
"AND table_id IN (SELECT tableVersions.table_id FROM version, tableVersions," + "
"WHERE version.id = tableVersions.version_id  " +
"AND tableVersions.table_id = tables.id  " +
"AND version.id= "+current+" "+
"AND tableVersions.table_id  " +
"IN ( SELECT tableversions.table_id FROM version, tableVersions, tables  " +
"WHERE version.id = tableVersions.version_id  " +
"AND tableVersions.table_id = tables.id  " +
"AND version.id= "+previous+")) " +
"UNION ALL (SELECT column_id, columntype.name,columns.name,id,minCard,maxCard," +
"type,length,decimalNumber,defaultValue,table_id FROM columntype, columns  " +
"WHERE columnType.column_id = columns.id  " +
"AND version_id="+current+" " +
"AND table_id IN (SELECT tableVersions.table_id FROM version, tableVersions," +
"tables  " +
"WHERE version.id = tableVersions.version_id  " +
"AND tableVersions.table_id = tables.id  " +
"AND version.id= "+current+" " +
"AND tableVersions.table_id  " +
"IN ( SELECT tableversions.table_id FROM version, tableVersions, tables  " +
"WHERE version.id = tableVersions.version_id  " +
"AND tableVersions.table_id = tables.id  " +
"AND version.id= "+previous+")))) t " +
"GROUP BY column_id, columntype_name,columns_name,id,minCard,maxCard," +
"type,length,decimalNumber,defaultValue,table_id " +
"HAVING COUNT(*) = 1;";
```

Le principe de cette requête mySQL repose sur la comparaison du résultat de deux clauses SELECT. S'il n'y a pas de différence, le résultat de la requête sera vide. S'il y a une différence, alors la requête retourne les tables dont une ou plusieurs colonnes ont étés modifiées.

Dans ces clauses, on récolte toutes les informations du nom et du typage des colonnes dans deux versions consécutives. Pour y parvenir, nous réalisons un UNION ALL du résultat des clauses SELECT. S'il existe des résultats qui sont présents dans une seule des deux versions (condition HAVING COUNT(*) = 1 du GROUP BY de toutes les colonnes) alors nous enregistrons les valeurs de **table_id** obtenues et nous

incrémentons de 1 la cellule des tables qui correspond au développeur responsable de la version *current*. L'ensemble de ces opérations est réalisé exclusivement pour les tables qui n'ont pas été ajoutées, supprimées, renommées, scindées entre les deux versions analysées.

3.3.6 Evaluation de notre méthode sur des cas pratiques

Des tableaux de relations "Une ou plusieurs colonne(s) de la table X a(ont) été modifiée(s) par le développeur Y" ont été obtenus par notre outil sur de véritables données historiques de systèmes existants. Un tableau de 278 tables (lignes) pour 25 développeurs (colonnes) a été obtenu pour le projet Broadleaf. Un tableau de 100 tables (lignes) pour 43 développeurs (colonnes) a été obtenu pour le projet OpenMRS.

Pour synthétiser l'information, la distribution des valeurs des tableaux de relations "Une ou plusieurs colonne(s) de la table X à(ont) été modifiée(s) par le développeur Y" pour les projets Broadleaf et OpenMRS sont reprises sous forme de diagramme en bâtons en figure 3.5.

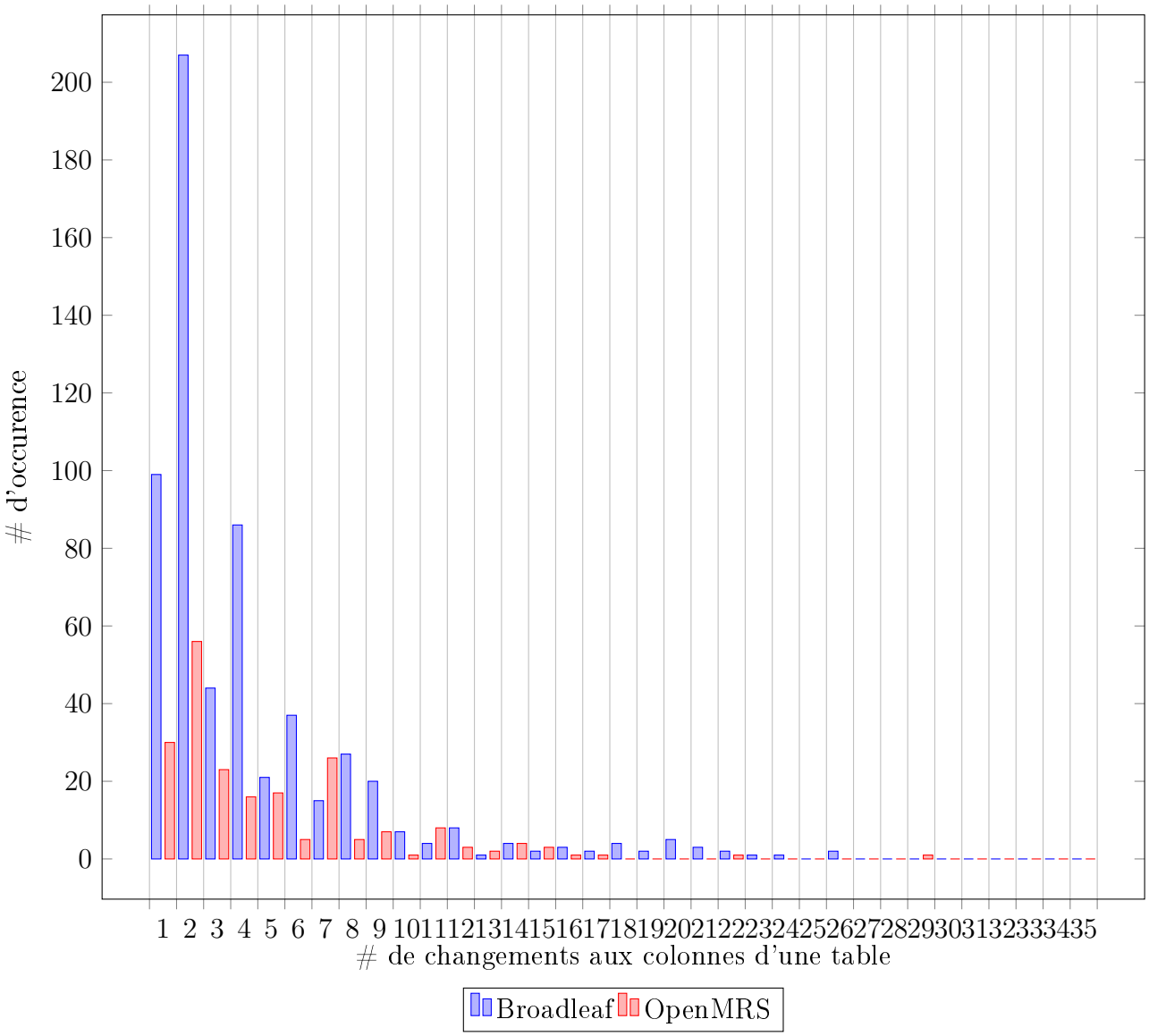
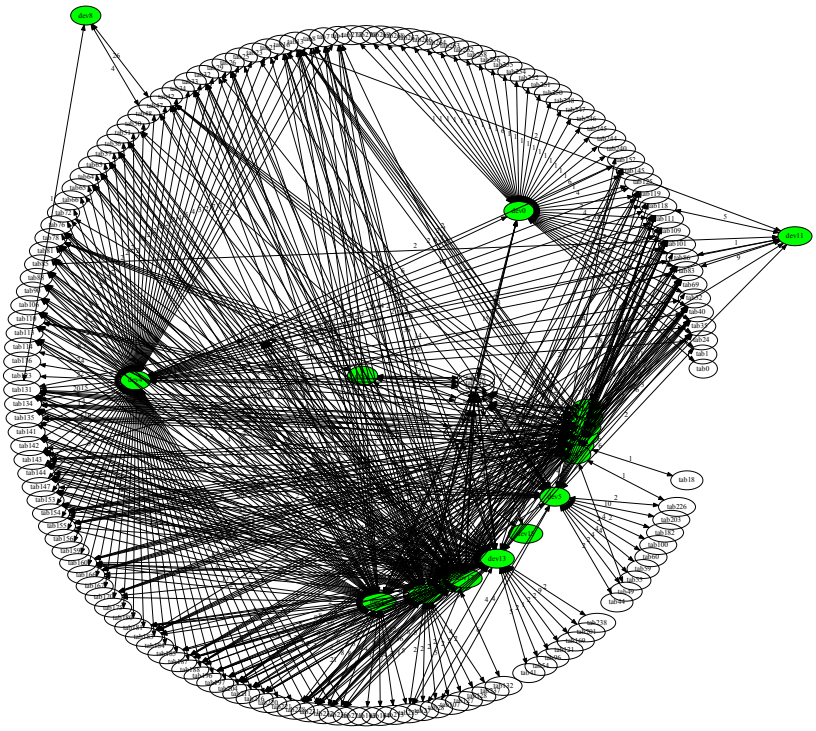
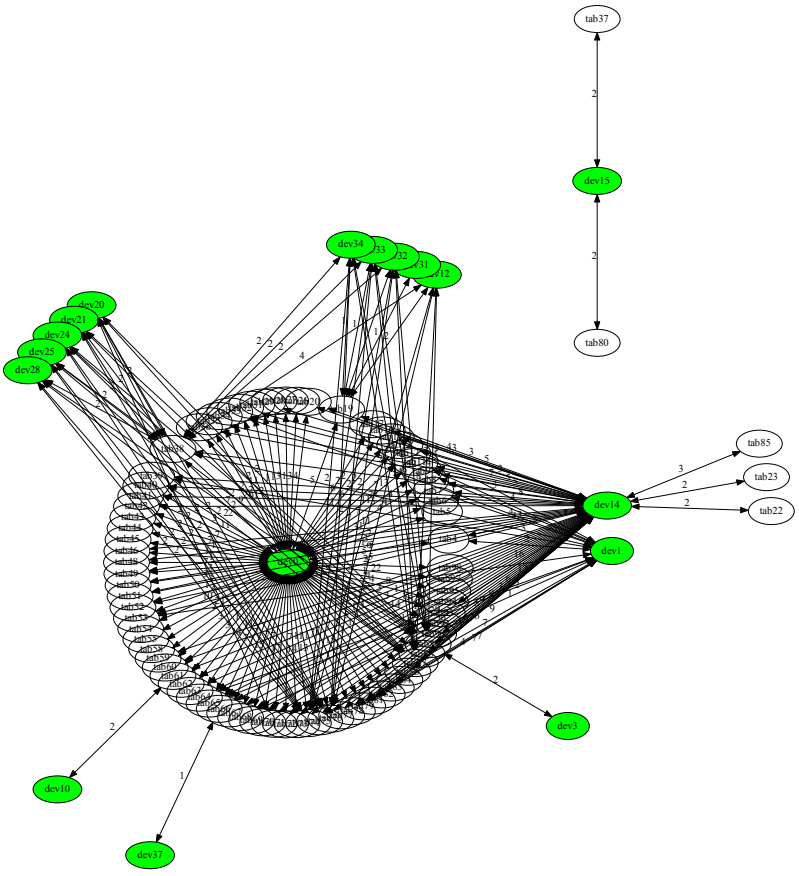


FIGURE 3.5 – Diagramme en bâtons de la distribution des valeurs des tableaux de relations "Une ou plusieurs colonne(s) de la table X à(ont) été modifiée(s) par le développeur Y"

Ces tableaux sont de bien trop grande taille que pour être inclus sous forme de texte. Nous les avons représentés sous forme de graphes pour les visualiser. Les tables et les développeurs sont les noeuds du graphe. Les relations "Une ou plusieurs colonne(s) de la table X à(ont) été modifiée(s) par le développeur Y" sont les arrêtes entre les noeuds. Le poids des arrêtes représente le nombre de changements à une ou plusieurs colonne(s) d'une table. De tels graphes sont représentés en figure 3.6a pour Broadleaf et en figure 3.6b pour OpenMRS.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.6 – Graphe des relations "Une ou plusieurs colonne(s) de la table X a(ont) été modifiée(s) par le développeur Y"

Le développeur X a modifié une ou plusieurs colonne(s) d'une même table que le développeur Y

Le tableau de relations "Une ou plusieurs colonne(s) de la table X a(ont) été modifiée(s) par le développeur Y" permet d'établir des relations entre développeurs. Les liens sont établis grâce aux changements que des développeurs ont apporté aux colonnes d'une ou plusieurs table(s) commune(s).

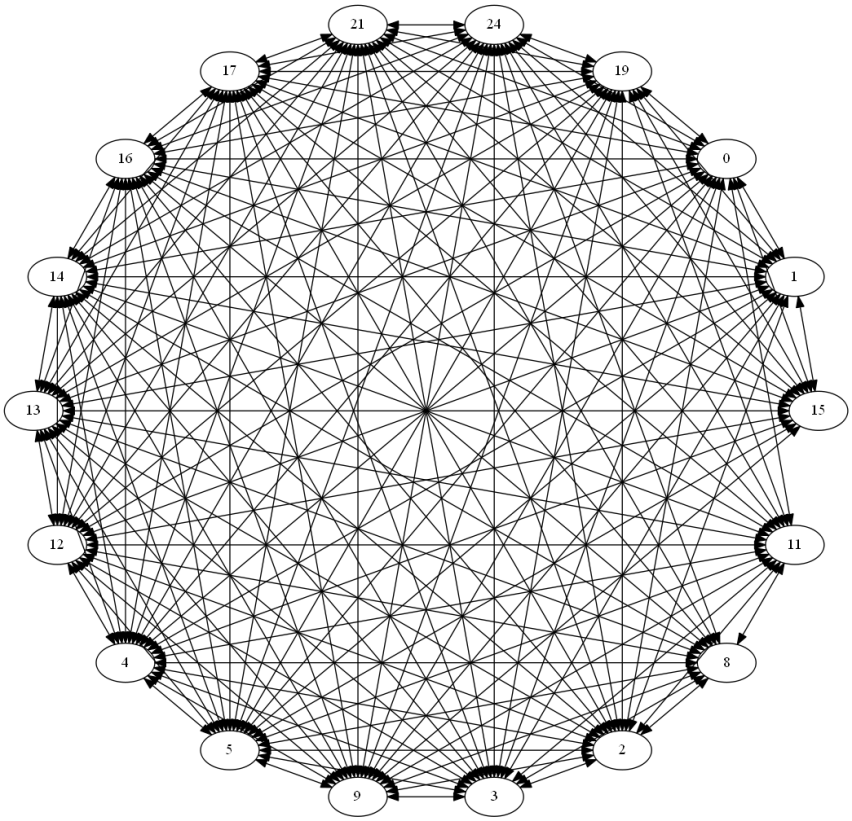
Pour Broadleaf , la table de relations est illustrée sous forme de graphe en figure 3.7a. On constate que, sur 18 développeurs actifs sur ce genre d'activités, 18 développeurs ont changé des colonnes dans une table où un autre développeur a lui aussi changé une colonne (100%). La connectivité du graphe est très forte.

Sur un total de 25 développeurs actifs sur le code, 18 ont donc touché au moins à une colonne d'une table du schéma. Cela veut dire que 72% des développeurs de Broadleaf apportent des modifications aux colonnes du schéma.

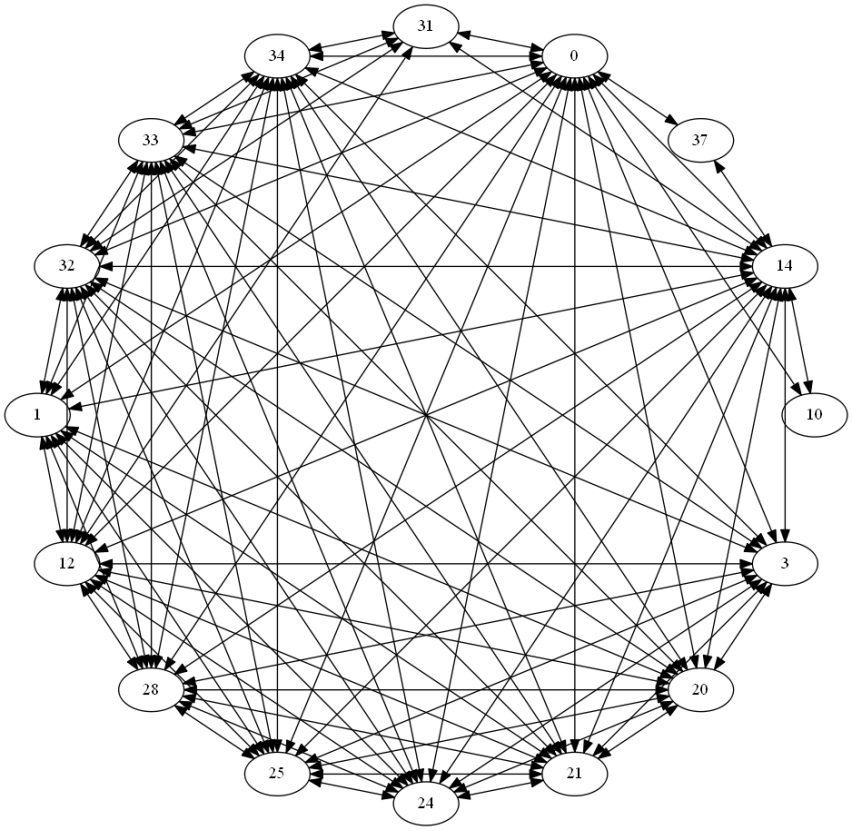
Pour OpenMRS , la table de relations est illustrée sous forme de graphe en figure 3.7b. On constate que, sur 17 développeurs actifs sur ce genre d'activités, 16 développeurs ont changé des colonnes dans une table où un autre développeur a lui aussi changé une colonne (94%). La connectivité du graphe est très forte.

Sur un total de 43 développeurs actifs sur le code, 16 ont donc touché au moins à une colonne d'une table du schéma. Cela veut dire que 37% des développeurs de OpenMRS apportent des modifications aux colonnes du schéma.

Globalement , dans le cas de la collaboration sur les modifications des colonnes, le contraste n'est plus aussi flagrant entre la communauté de développeurs de Broadleaf et de OpenMRS. Seule la proportion de développeurs actifs sur le schéma est différente entre les deux projets (72% pour Broadleaf contre 37% pour OpenMRS). La séparation des résponsabilités entre le code et le schéma de la base de données semble plus forte dans le cas de OpenMRS.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.7 – Graphe des relations "Développeur X a modifié une ou plusieurs colonne(s) d'une même table que le développeur Y"

3.3.7 Améliorations possibles

Nous établissons des liens entre développeurs s'ils ont tous les deux apporté au moins un changement à une colonne d'une table commune. Dans le cas où cette méthode engendrerait des *faux positifs*, nous proposons des pistes d'amélioration :

— Tout comme l'analyse des modifications apportées aux tables, il serait possible d'affiner les résultats en imposant un plus grand nombre de tables communes sur lesquelles des modifications ont été apportées pour établir un lien entre deux développeurs.

— Contrairement à l'analyse sur les modifications apportées aux tables, le nombre de changements possibles apportés aux colonnes d'une table est plus élevé. Ceci est dû au fait qu'une table peut continuer à exister pendant que l'on ajoute, supprime, renomme ou change le type d'une ou plusieurs de ces colonnes. Il est donc possible d'affiner les résultats en choisissant un nombre de changements >1 apporté à une même table.

Pour évaluer l'influence de ce paramètre, nous avons appliqué cette méthode pour un nombre de colonnes n compris entre >0 et >14. Les résultats suivants sont repris dans le tableau 3.7. :

— le nombre de relations entre les développeurs (nombre d'arrêtes dans le graphe) ;

— le nombre de développeurs dans un réseau (nombre de noeuds dans le graphe) ;

— le pourcentage du nombre de relations entre les développeurs pour >n colonnes comparé au maximum obtenus pour >0 colonne ;

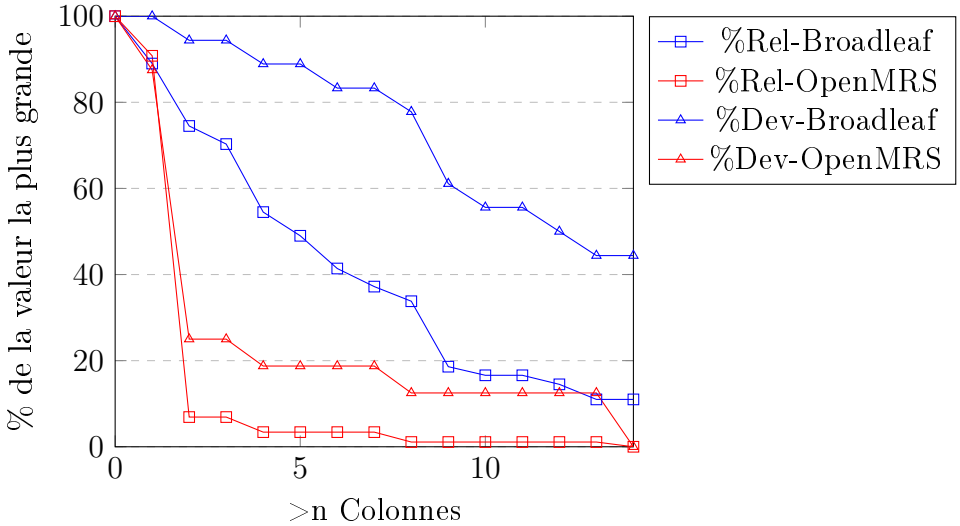
— le pourcentage du nombre de développeurs dans le réseau pour >n colonnes comparé au maximum de développeurs dans le réseau pour >0 colonne.

Les valeurs des pourcentages en fonction de n sont illustrées sous forme de courbes en figure 3.8.

TABLE 3.7 – Nombre de relations entre les développeurs et Nombre de développeurs dans le réseau en fonction du paramètre n colonnes communes

>n	Broadleaf				OpenMRS			
	#rel	#dev	%relMax	%devMax	#rel	#dev	%relMax	%devMax
0	145	18	100,0	100,0	87	16	100	100,0
1	129	18	89,0	100,0	79	14	90,8	87,5
2	108	17	74,5	94,4	6	4	6,9	25,0
3	102	17	70,3	94,4	6	4	6,9	25,0
4	79	16	54,5	88,9	3	3	3,4	18,8
5	71	16	49,0	88,9	3	3	3,4	18,8
6	60	15	41,4	83,3	3	3	3,4	18,8
7	54	15	37,2	83,3	3	3	3,4	18,8
8	49	14	33,8	77,8	1	2	1,1	12,5
9	27	11	18,6	61,1	1	2	1,1	12,5
10	24	10	16,6	55,6	1	2	1,1	12,5
11	24	10	16,6	55,6	1	2	1,1	12,5
12	21	9	14,5	50,0	1	2	1,1	12,5
13	16	8	11,0	44,4	1	2	1,1	12,5
14	16	8	11,0	44,4	0	0	0	0,0

FIGURE 3.8 – Pourcentage du nombre de relations pour n colonnes comparé au maximum obtenu pour 0 colonne



Pour les valeurs obtenues avec les conditions n>0 colonne et n>1 colonne, la diminition du pourcentage de relations est équivalente pour les deux projets. Ensuite, nous constatons que dans le cas de OpenMRS, une valeur de n>2 fait diminuer drastiquement le nombre de relations détectées. Dans le cas de Broadleaf, le nombre de relations diminue beaucoup moins rapidement en fonction de n.

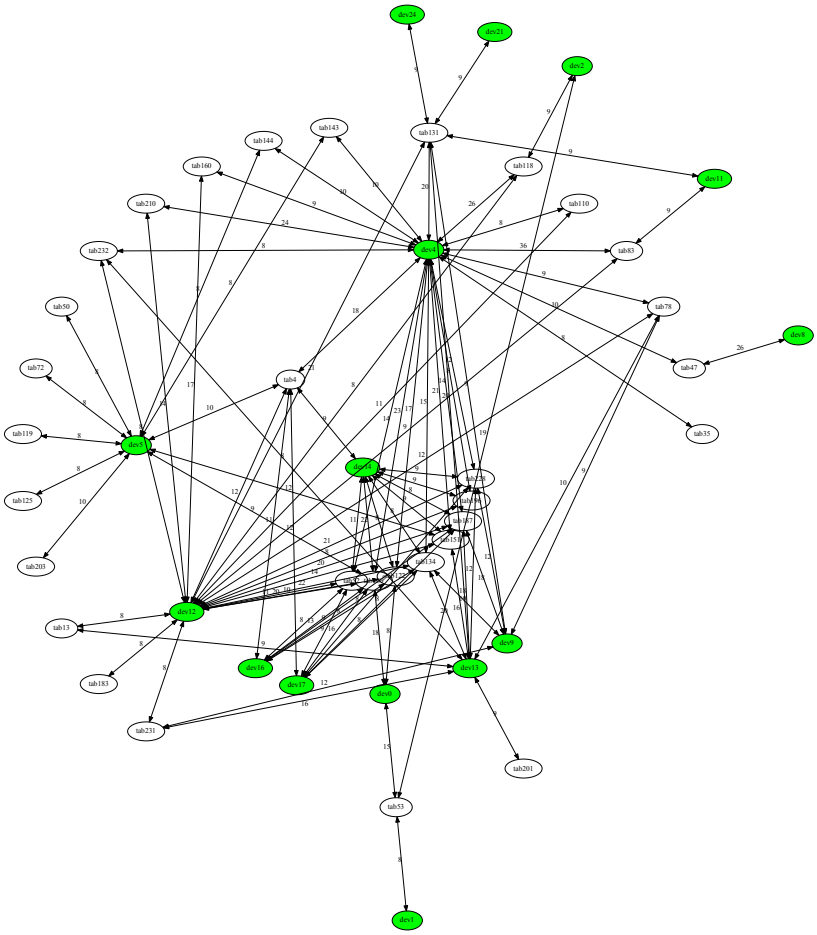
La valeur n devrait être choisie dans le but de diminuer la détection des faux

positifs, tout en conservant une bonne sensibilité. Ce nombre pourrait être choisi objectivement si nous avions une référence à laquelle comparer nos résultats. Les résultats obtenus par la question de recherche 5 pourraient toutefois servir de référence.

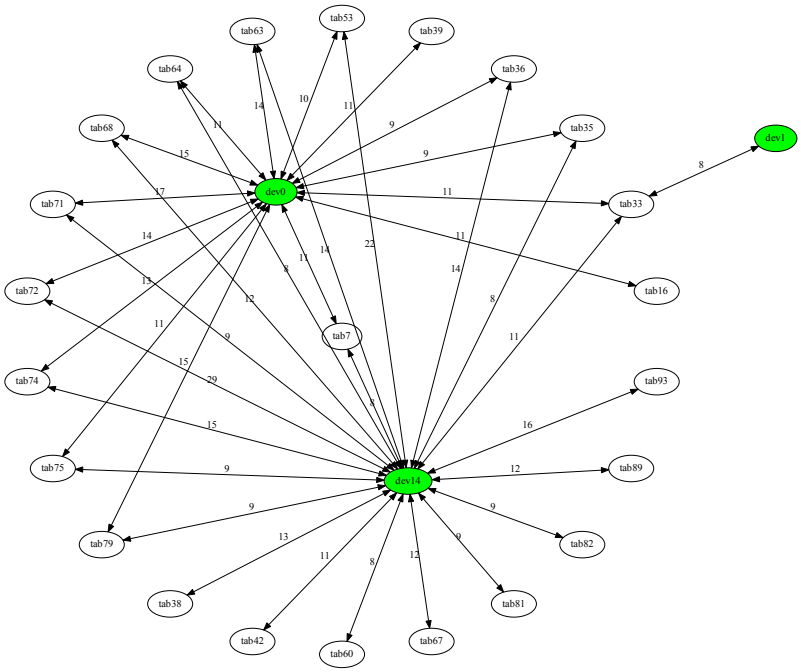
Néanmoins, au vu de la forte décroissance des résultats obtenus dans le cas de OpenMRS, ce paramètre pourrait éventuellement diminuer trop fortement la sensibilité de la détection et donc réduire la détection des vrais positifs (augmenter les faux négatifs).

Nous avons choisi arbitrairement une valeur $n > 7$ pour illustrer le principe. Nous en présentons le résultat dans les graphes de relations tables-développeurs en figure 3.9a pour Broadleaf, et en figure 3.9b pour OpenMRS.

A partir de ces graphes, sont ensuite dérivés les graphes de relations entre développeurs illustrés en figure 3.10a pour Broadleaf et 3.10b pour OpenMRS.

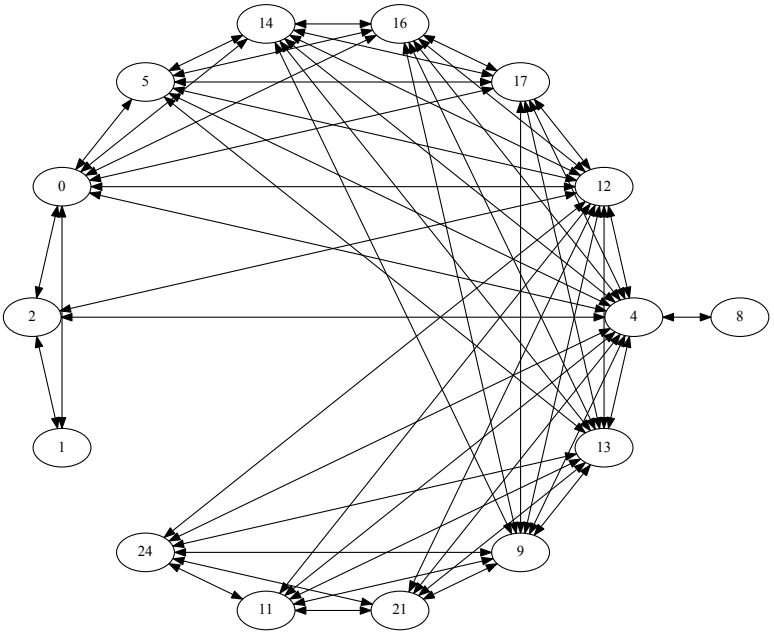


(a) Broadleaf

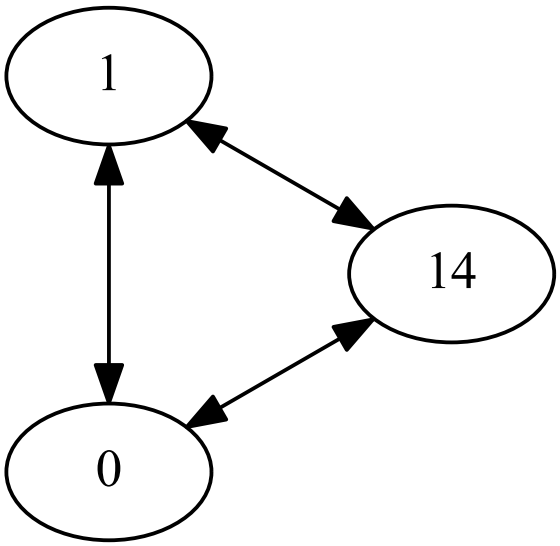


(b) OpenMRS

FIGURE 3.9 – Graphe des relations "Le développeur X a modifié plus de 7 colonnes d'une table Y"



(a) Broadleaf



(b) OpenMRS

FIGURE 3.10 – Graphe des relations "Les développeurs X et Y ont modifié plus de 7 colonnes d'une table en commun "

3.4 Question de recherche 2 : Peut-on détecter les développeurs ayant effectué des changements dans le code du programme sur des appels communs à la base de données ?

Nous avons vu au chapitre 1, que [De Souza, Froehlich et Dourish 2005],[Martinez-Romo et al. 2008] ont déja réalisé des travaux pour établir des liens entre des développeurs en se basant sur des modifications à un artefact technique commun tel que le code. Nous souhaitons proposer plusieurs pistes pour élargir leurs travaux, non seulement au niveau du code, mais aussi au niveau des appels à la base de données dans le code.

3.4.1 Données disponibles dans le schéma conceptuel de [Meurice et al. 2016]

Nous pensons que les données fournies par l'outil **DA**tabase sc**H**ema evo**L**ut**I**on **A**nalysis (DAHLIA) qui sont décrites par le schéma conceptuel de [Meurice et al. 2016] contiennent suffisement d'informations pour établir des liens entre développeurs jusqu'au niveau des appels à la base de données.

Au moyen des données décrites par ce schéma conceptuel en figure 2.4, et plus particulièrement les parties **DatabaseAcess** en figure 2.9, **CodeObject** en figure 2.7 et **Mapping** en figure 2.10, nous pensons qu'il est possible de construire des réseaux comme ceux obtenus à la question de recherche 1.

Nous pensons qu'il sera possible d'établir des liens entre les développeurs grâce aux informations suivantes :

- les artefacts techniques tel que le code source, en utilisant la partie CodeObject du schéma conceptuel ;
- le code source en rapport ou non avec les appels à la base de données, en utilisant les données disponibles dans les parties CodeObject et DatabaseAccess ;
- les technologies communes d'accès à la base de données, en utilisant les attributs de DatabaseAccess.

3.4.2 Méthodologie

Nous pouvons généraliser notre approche utilisée en question 1 par les étapes suivantes :

1. **détecter les changements qu'un développeur apporte à un type d'artefact technique dans une version n** en comparant celle-ci à la version n-1 ;
2. **établir un réseau "Développeurs-Artefact Technique"** ;
3. **établir un réseau "développeurs-développeurs"** au travers des liens entre eux et les artefacts techniques communs.

Similairement à la question 1, les réseaux obtenus aux étapes 2 et 3 sont alors analysables qualitativement (visualisation des réseaux) ou quantitativement (dérivation d'informations supplémentaires à partir des données du réseau comme les noeuds, arrêtes et le poids de celles-ci).

Les étapes 2 et 3 ont déja été implémentées pour répondre à la question 1 et peuvent être réutilisées. Par contre, l'étape de détection nécessite d'établir la requête SQL qui comparera correctement les deux versions. Ce point n'a pas été développé dans ce travail mais mériterait toutefois d'être exploré de manière approfondie.

3.4.3 Resultats escomptés

La plus value de ces méthodes d'analyse des réseaux sociaux dans le domaine des Data-Intensive Software Systems (DISS) a été abordée en fin du chapitre 1 dans la table 2.1. Nous pensons que les méthodes de Social Network Analysis (SNA) utilisées par [Martinez-Romo et al. 2008] permettraient d'analyser quantitativement des paramètres de collaboration entre les développeurs tel que le degré de coordination d'une équipe.

Notre but initial était d'obtenir des réseaux de liens entre les développeurs au niveau du schéma de la base de données (Question 1). Ensuite, d'établir des liens au niveau du code accédant à la base de données (Question 2) pour finalement établir des liens entre ces deux réseaux (Question 4) afin de fournir des informations permettant potentiellement d'objectiver les similaritées entre ces deux réseaux. Notre intuition est en effet que ces résultats pourraient apporter des informations sur les efforts collaboratifs de co-évolution schéma-code.

3.4.4 Amélioration possible

Au moyen du réseau "développeurs-artefact technique", nous dérivons des réseaux "développeurs-développeurs". Il est interessant de constater que l'inverse est possible et que nous pouvons obtenir des relation "artefact techniques-artefact techniques".

3.5 Question de recherche 3 : Peut-on déterminer quels développeurs ont été contemporains ?

Pour tous les développeurs appartenant au projet, nous voulons démontrer l'existence ou l'absence d'une relation "est contemporain" entre deux développeurs. Notre but est d'établir si les développeurs ont en commun une période de temps pendant laquelle ils ont tous les deux été actifs sur le projet. Ceci permettrait d'enrichir les résultats des questions de recherche 1 et 2. En effet, si deux développeurs modifient un artefact technique commun et sont contemporains, il est possible qu'ils aient collaboré. Au contraire, s'ils n'ont pas été contemporains et ont modifié un artefact technique commun, alors on peut imaginer qu'un développeur a du apporter la modification sans pouvoir consulter le développeur qui a quitté le projet (turnover).

3.5.1 Méthodologie

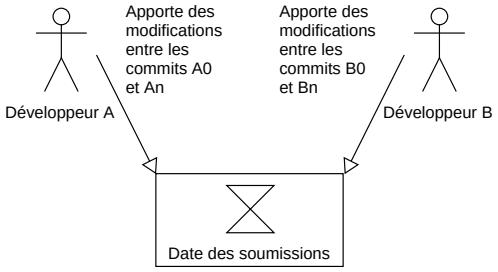


FIGURE 3.11 – Lien entre des développeurs au travers d'une période d'activité commune.

Notre objectif est d'obtenir un tableau à deux dimensions égales. Les lignes et les colonnes correspondent aux développeurs travaillant sur le projet. Deux groupes de développeurs seront distingués selon la valeur de la cellule commune aux deux developpeurs :

— si la valeur est à 1, les développeurs ont été contemporains. Ceci est détecté par un chevauchement des soumissions des deux développeurs ;

— si la valeur est à 0, les développeurs n'ont pas été contemporains. Il n'y a donc pas de chevauchement entre les soumissions des deux développeurs.

Un exemple de tableau de relation "est contemporain de" est illustré par la table 3.8 .

Pour construire ce tableau, nous nous basons sur la liste historique des versions et des développeurs responsables de celles-ci.

TABLE 3.8 – Matrice de relations "le développeur X est contemporain au développeur Y"

Dx/Dy	D0	D1	D2
D0	0	1	0
D1	1	0	0
D2	0	0	0

Liste historique des versions et des développeurs responsables de celles-ci

La liste historique des versions est un tableau contenant dans l'ordre chronologique des soumissions, les identifiants des développeurs qui ont soumi chaque version. L'identifiant est l'adresse mail du développeur. Nous l'obtenons avec la requête mySQL suivante :

```
SELECT id, developer FROM version;
```

Un exemple d'historique est illustré par la table 3.9 où nous pouvons observer que :

— D1, a effectué un commit entre plusieurs commits de D0. On considère donc que D1 et D0 sont contemporains. Les cellules en commun aux deux développeurs seront mises à la valeur 1 ;

— D2, n'a effectué aucun commit entre les commits des développeurs D0 et D1. Nous considérons donc que D2 n'est pas contemporain avec D0 ni D1. Les cellules communes entre ces développeurs restent à la valeur 0.

TABLE 3.9 – Liste historique des versions et des développeurs reponsables de celles-ci

id Version	id du développeur
0	D0
1	D0
2	D1
3	D0
4	D1
5	D2
6	D2
n	...

3.5.2 Evaluation de notre méthode sur des cas pratiques

Des tableaux de relations ont été obtenus par notre outil sur de véritables données historiques de systèmes existants. Ils sont repris en table 3.10 pour Broadleaf et en table 3.11 pour OpenMRS.

TABLE 3.10 – Tableau de relations "le développeur X est contemporain au développeur Y" pour Broadleaf

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
5	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
11	0	0	0	0	1	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0
12	0	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0
13	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	0
14	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
15	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0
17	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0
18	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE 3.11 – Tableau de relations "le développeur X est contemporain au développeur Y" pour OpenMRS

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
13	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
15	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
16	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
21	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

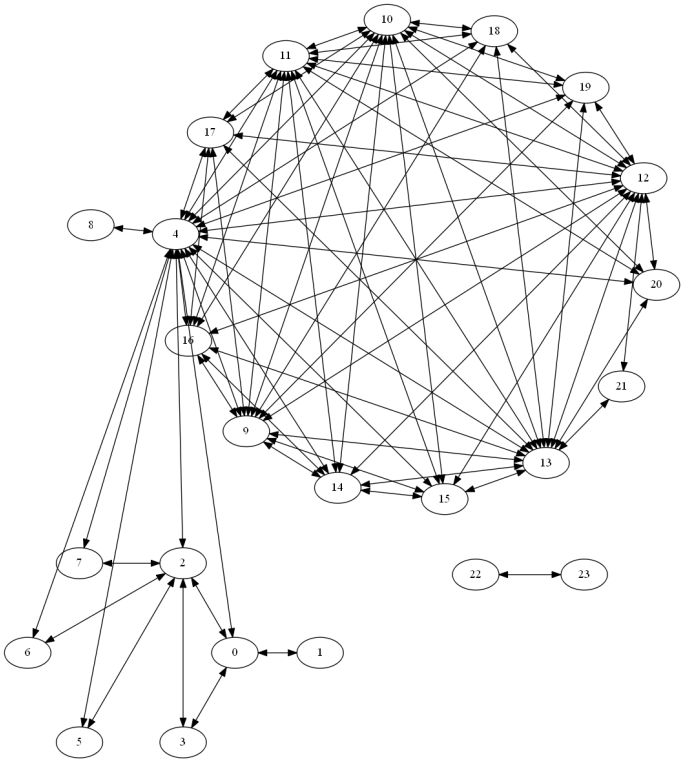
Ces tableaux peuvent aussi être présentés sous forme de graphes. Les développeurs sont les noeuds du graphe, les relations "X est contemporain de Y" sont les arrêtes entre les noeuds. De tels graphes sont représentés en figure 3.12a pour Broadleaf et en figure 3.12b pour OpenMRS.

Pour Broadleaf , on constate visuellement avec le graphe en figure 3.12a que :

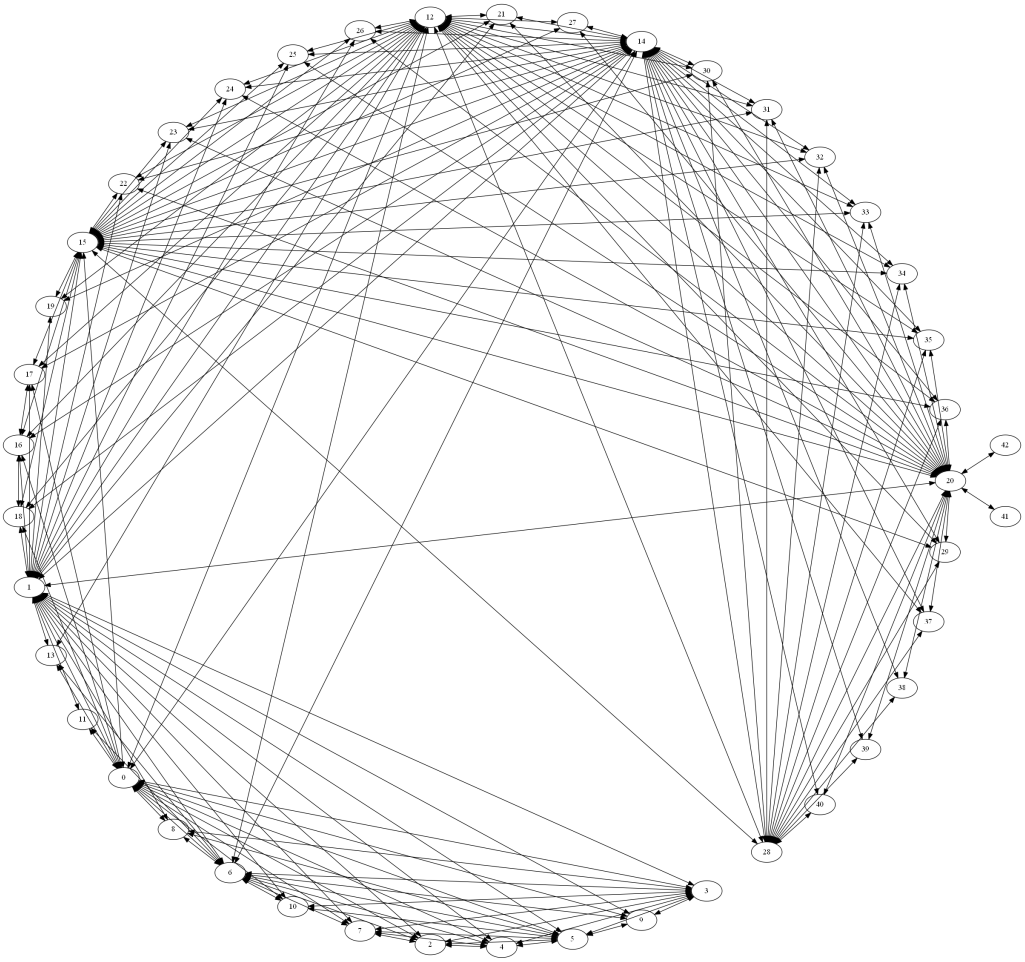
- une équipe initiale est contemporaine $\{0, 1, 2, 3, 4, 5, 6, 7\}$ et seul le développeur 4 de l'équipe initiale est contemporain du reste des développeurs. Il y a peut-être eu un important turn-over à un moment du projet. La connectivité de ce sous-graphe est faible ;
- une deuxième équipe est composée de $\{4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21\}$. La connectivité de ce sous-graphe est plus forte ;
- les développeurs 22 et 23 sont les derniers arrivés dans l'équipe et ne sont pas encore contemporains avec la deuxième équipe. Il est peut-être trop tôt pour constater un deuxième turn-over de l'équipe ;
- le développeur 8 n'as pas travaillé longtemps sur le projet car il a seulement eu le temps d'être contemporain avec le développeur 4.

Pour OpenMRS , on constate visuellement avec le graphe en figure 3.12a que :

- la connectivité du graphe est plus grande que dans le cas de Broadleaf. C'est-à-dire qu'il y a un peu de turn-over, mais il y a moins de noeuds critiques qui doivent disparaitre pour déconnecter le graphe ;
- les développeurs $41, 42$ sont seulement contemporains d'un autre développeur (20). Ils sont les derniers arrivés et il est peut-être trop tôt pour tirer des conclusions à leur sujet.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.12 – Graphe de relations "le développeur X est contemporain au développeur Y"

3.5.3 Collaboration ou turnover ?

Collaboration car contemporain

Pour déterminer si les relations "Développeur X a modifié une même table du schéma que le développeur Y" pourraient être de réelles relations de collaboration, il faut exclure les liens entre des développeurs qui ne sont pas contemporains.

Ceci revient à combiner les réseaux en figure 3.4 ou 3.7 et le réseau en figure 3.12 au moyen d'un ET logique comme décrit en figure 3.13. On obtient alors un graphe des développeurs qui ont modifié un artefact technique commun ET qui sont contemporains.

FIGURE 3.13 – Utilisation d'un ET logique pour obtenir un réseau de développeurs contemporains

Turnover car non-contemporains

Pour déterminer si les relations "Développeur X a modifié une même table du schéma que le développeur Y" sont plutôt dues à du turnover, il faut exclure les liens entre des développeurs qui ne sont pas contemporains.

Ceci revient à combiner les réseaux en figure 3.4 ou 3.7 et le complément du réseau en figure 3.12 au moyen d'un ET et d'un NON logiques comme décrit en figure 3.14. On obtient alors un graphe des développeurs qui ont modifié un artefact technique commun ET qui sont NON contemporains.

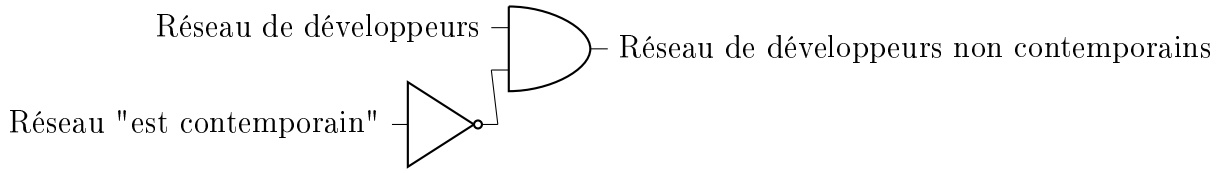


FIGURE 3.14 – Utilisation d'un NON et d'un ET logiques pour obtenir un réseau de développeurs non contemporains

3.5.4 Evaluation de notre méthode sur des cas pratiques

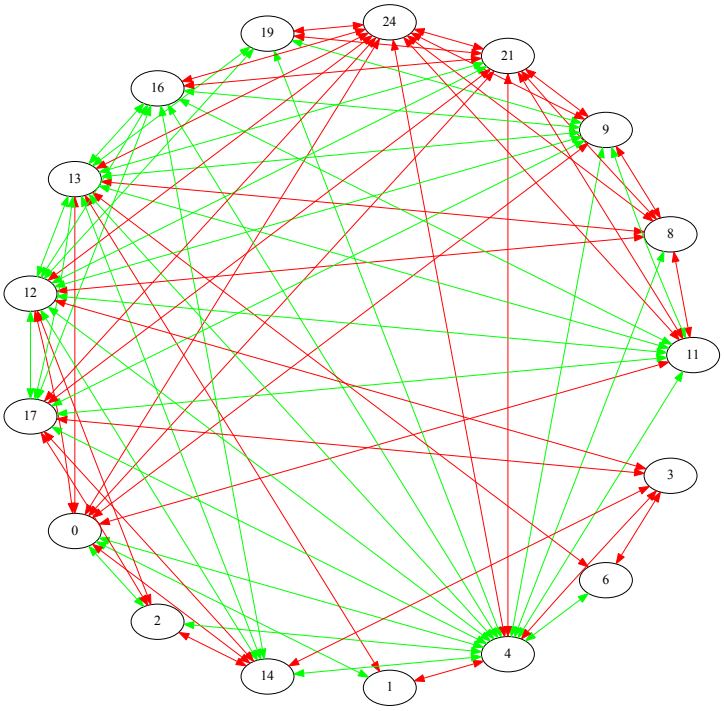
Résultats pour les changements apportés aux tables

Nous avons obtenu en section 3.3.2 des graphes de relations "le développeur X a changé une même table que le développeur Y". Nous utilisons l'information "est contemporain ou non" pour colorier les arrêtes du graphe :

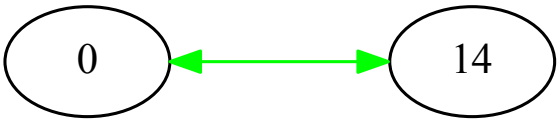
— les arrêtes sont vertes quand les deux développeurs ont changé une même table ET sont contemporains ;

— les arrêtes sont rouges quand les deux développeurs ont changé une même table ET ne sont pas contemporains.

Les résultats sont représentés en figure 3.15a pour Broadleaf et en figure 3.15b pour OpenMRS.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.15 – Graphe de relations "le développeur X a changé une même table que le développeur Y". En vert les développeurs étaient contemporains, en rouge les développeurs n'étaient pas contemporains.

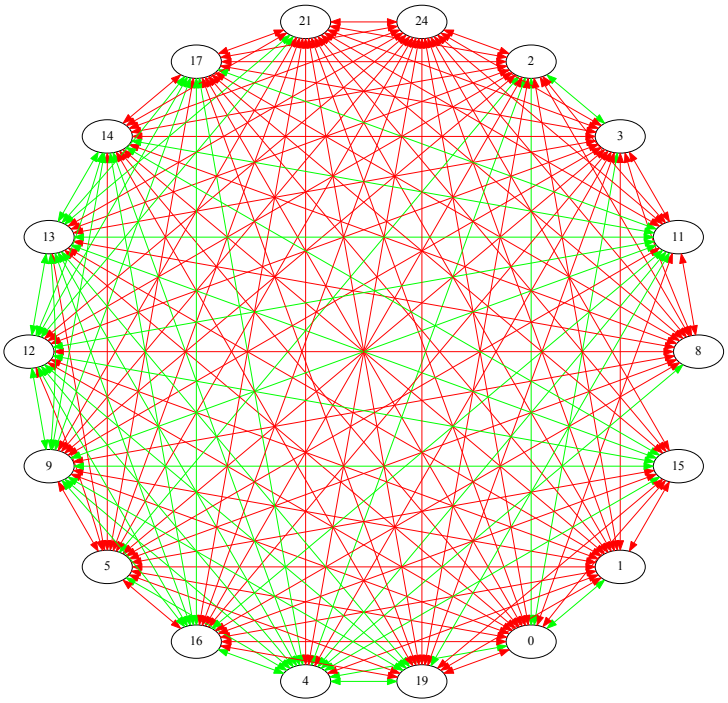
Résultats pour les changements apportés aux colonnes

En section 3.3.5, nous avons obtenu des graphes de relations "le développeur X a modifié une ou plusieurs colonne(s) d'une même table que le développeur Y". Nous utilisons l'information "est contemporain ou non" pour colorier les arrêtes du graphe :

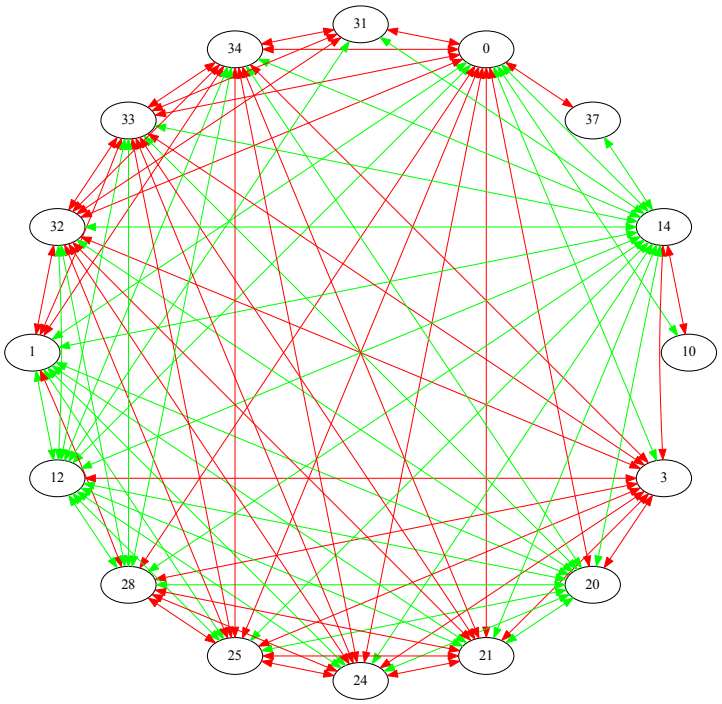
- les arrêtes sont vertes quand les deux développeurs ont changé une ou plusieurs colonne(s) d'une même table ET sont contemporains ;
- les arrêtes sont rouges quand les deux développeurs ont changé une ou plusieurs colonne(s) d'une même table ET ne sont pas contemporains.

Les résultats sont représentés en figure 3.16a pour Broadleaf et en figure 3.16b pour OpenMRS.

Dans le cas de Broadleaf, il y a 48 relations où les développeurs sont contemporains, 97 relations où les développeurs ne sont non pas contemporains. Dans le cas de OpenMRS, 41 relations sont contemporaines et 46 sont non contemporaines.



(a) Broadleaf



(b) OpenMRS

FIGURE 3.16 – Graphe de relations "le développeur X a modifié une ou plusieurs colonne(s) d'une même table que le développeur Y". En vert les développeurs sont contemporains, en rouge les développeurs ne sont pas contemporains.

Résultats pour les changements aux x colonnes

Comme discuté lors de la question de recherche 1, pouvoir détecter une véritable collaboration au travers d'une modification d'un artefact technique pourrait engendrer des **faux positifs**, c'est-à-dire que l'on détecterait un lien de collaboration entre deux développpeurs alors qu'il n'existe pas. Ceci peut être dû à un manque de communication ou tout simplement au turnover de l'équipe. Nous avons proposé d'augmenter la spécificité du test au moyen du nombre de colonnes modifiées dans une table commune pour établir un lien entre deux développeurs. Dans ce cas-ci, nous ré-évaluons les valeurs de la table 3.7 au moyen de l'information "est contemporain". Nous faisons état des résultats suivants dans la table 3.12 :

1. le nombre total de développeurs dans le graphe ;
2. le nombre total de développeurs connectés par des relations "est contemporain" ;
3. le nombre total de développeurs connectés par des relations "nest pas contemporain" ;
4. le nombre de relations "est contemporain" ;
5. le nombre de relations "n'est pas contemporain" ;
6. le pourcentage des relations "est contemporain" du total des relations entre les développeurs (item 4+ item 5) ;
7. le pourcentage des relations "est contemporain" du maximum (valeur obtenue pour un nombre de colonnes >0) ;
8. le pourcentage des relations "n'est pas contemporain" du maximum (valeur obtenue pour un nombre de colonnes >0).

Il apparait que peu importe n, le nombre de développeurs connectés dans le graphe par des arrêtes "est contemporain" est fortement équivalent au nombre total de développeurs dans le graphe. Par contre, plus la valeur de n colonnes augmente, plus le nombre de relations diminue. Ce phénomène est plus prononcé pour OpenMRS. Ceci est dû au fait que seuls quatres développeurs ont effectué plus de deux changements aux colonnes des tables.

TABLE 3.12 – Combinaison "est contemporain" avec "Le développeur X a modifié plus de n colonnes d'une table Y"

>n				Broadleaf				
	#Dev	Contemporain		#Rel Contemporain				
		Oui	Non	Oui	Non	%total	% max Oui	% Max Non
0	18	17	18	48	97	33,1	100,0	100,0
1	18	17	18	47	82	36,4	97,9	84,5
2	17	16	17	45	63	41,7	93,8	64,9
3	17	16	17	45	57	44,1	93,8	58,8
4	16	15	15	38	41	48,1	79,2	42,3
5	16	15	15	36	35	50,7	75,0	36,1
6	15	14	14	32	28	53,3	66,7	28,9
7	15	14	14	32	22	59,3	66,7	22,7
8	14	13	12	29	20	59,2	60,4	20,6
9	11	11	7	18	9	66,7	37,5	9,3
10	10	10	7	17	7	70,8	35,4	7,2
11	10	10	7	17	7	70,8	35,4	7,2
12	9	9	5	16	5	76,2	33,3	5,2
13	8	8	4	12	4	75,0	25,0	4,1
14	8	8	4	12	4	75,0	25,0	4,1
>n				OpenMRS				
	#Dev	Contemporain		#Rel Contemporain				
		Oui	Non	Oui	Non	%total	% max Oui	% Max Non
0	16	16	16	41	46	47,1	100,0	100,0
1	14	14	14	38	41	48,1	92,7	89,1
2	4	4	0	6	0	100,0	14,6	0,0
3	4	4	0	6	0	100,0	14,6	0,0
4	3	3	0	3	0	100,0	7,3	0,0
5	3	3	0	3	0	100,0	7,3	0,0
6	3	3	0	3	0	100,0	7,3	0,0
7	3	3	0	3	0	100,0	7,3	0,0
8	2	2	0	1	0	100,0	2,4	0,0
9	2	2	0	1	0	100,0	2,4	0
10	2	2	0	1	0	100,0	2,4	0
11	2	2	0	1	0	100,0	2,4	0
12	2	2	0	1	0	100,0	2,4	0
13	2	2	0	1	0	100,0	2,4	0
14	0	0	0	0	0	NaN	0	0

3.5.5 Conclusion

On constate qu'il est important de faire la différence entre les relations établies lorsque les développeurs sont contemporains ou non. Pour les deux systèmes évalués, nous avons pu observer que seuls 33% à 47% des modifications apportées aux colonnes du schéma sont faites par des développeurs qui étaient contemporains.

Cela signifie que les développeurs modifient tout autant des tables qu'ils soient contemporains ou non avec l'auteur de la table. Et donc qu'ils ne collaborent pas nécessairement au moyen de communications formelles mais par d'autres canaux tels que la documentation, le code lui-même, etc.

On constate aussi que lorsque nous filtrons les relations entre développeurs sur une fenêtre de temps, cela permet de conserver un plus grand nombre de développeurs en relation entre eux que d'appliquer une contrainte d'un nombre de changements sur un objet commun.

Dans le cas de Broadleaf, on conserve 17 développeurs sur les 18 en relation si l'on ignore les relations entre développeurs non-contemporains. Dans le cas de OpenMRS, on conserve 16 développeurs sur 16 en relation entre eux.

Si l'on impose un nombre de modifications avant de considérer un lien entre des développeurs, les résultats sont très différents entre Broadleaf et OpenMRS. Le réseau de développeurs de OpenMRS diminue abruptement de 16 à 4 noeuds en imposant plus de 2 modifications de colonnes à une table commune avant de faire un lien entre des développeurs. Cette méthode peut par contre faire la différence entre les développeurs qui font des changements précis et ponctuels et ceux qui font des changements plus importants ou plus fréquents.

Globalement, si nous n'utilisons pas au moins une forme de filtre temporel, le phénomène de turn-over influencerait trop les résultats et ne premettrait pas d'établir des liens précis entre des développeurs.

3.5.6 Améliorations possibles

Cette méthode est applicable pour filtrer les graphes de collaboration sur toute la durée d'un projet, où un important turn-over qui pourrait avoir beaucoup d'influence sur des résultats qui n'auraient pas été filtrés.

Une possibilité d'amélioration serait de quantifier la durée de temps commune (intersection des dates des commits entre deux développeurs). Cette durée de temps commune peut être exprimée soit au nombre de jours soit au nombre de commits pendant cette durée. Cette dernière peut être utilisée pour donner un poids à l'arrête reliant les deux développeurs. L'intuition étant qu'une longue période de travail commune sur un projet au cours de laquelle des objets communs ont été modifiés serait significative d'un travail en équipe.

Une autre possibilité d'amélioration serait d'obtenir les graphes issus des questions de recherche 1 et 2 mais d'en limiter l'observation pendant la durée de l'exécution d'une tâche. Pour cela, il faudrait faire les liens entre les développeurs ayant participé à une tâche en particulier et considérer les soumissions comprises entre la date de début et la date de fin de la planification de cette tâche. Les informations nécessaires ne sont pas disponibles dans le schéma conceptuel mais peuvent être obtenues au moyen des données sauvegardées par un système de planification des tâches.

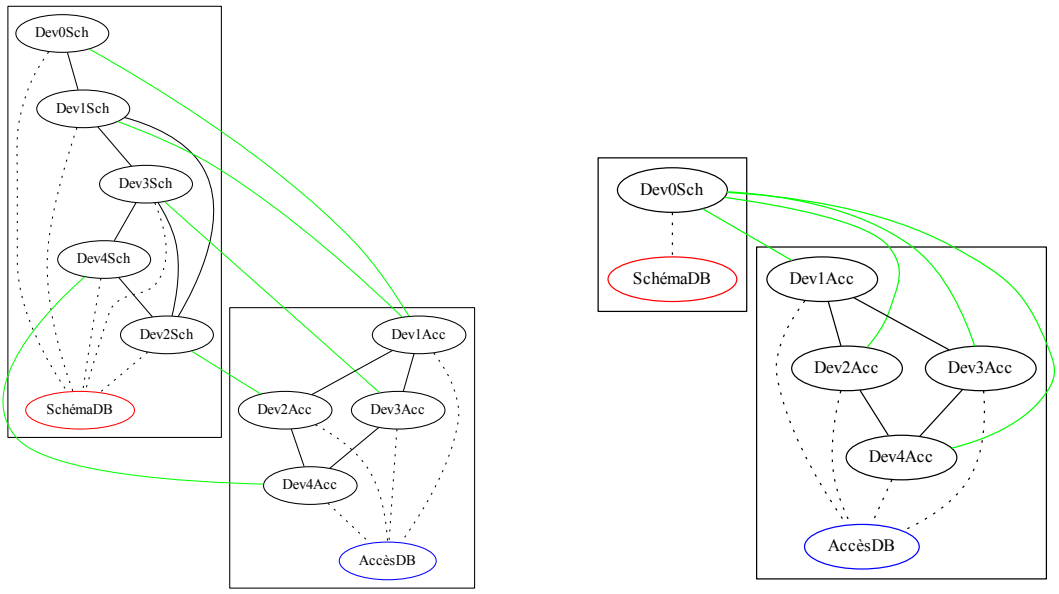
3.6 Question de recherche 4 : Les comportements collaboratifs repris aux questions 1 et 2 sont-ils semblables ?

3.6.1 Méthodologie

Répondre à cette question nécessite d'utiliser une méthode telle que décrite par [Valetto et al. 2007] pour évaluer la congruence entre les deux réseaux. Cette méthode, appliquée à notre cas de figure, permettrait d'évaluer le dégré d'alignement des relations entre développeurs au niveau du code avec les relations entre développeurs au niveau du schéma. Nous proposons une piste de réflexion sur les moyens à mettre en oeuvre pour répondre à cette question.

Nous décrivons la démarche nécessaire au moyen des étapes suivantes :

1. **construction du réseau de développeurs relatif au schéma de la DB.** (Réponse à la question de recherche 1). Une modélisation de résultats possibles est illustrée en figure 3.17. Il s'agit du sous-graphe **SchémaDB**. Le réseau de développeurs de ce sous-graphe est construit via les objets du schéma de la base de données représentés par les arrêtes pointillées ;
2. **construction du réseau de développeurs relatif aux accès à la DB.** (Réponse à la question de recherche 2). Une modélisation de résultats possibles est illustrée en figure 3.17. Il s'agit du sous-graphe **AccèsDB**. Le réseau de développeurs de ce sous-graphe est construit via les accès à la base de données représentés par les arrêtes pointillées ;
3. **établir des liens entre les deux réseaux** Pour cela, il faut utiliser les liens qui existent entre les accès à la DB et les databases objects tels que décrits dans le schéma conceptuel de [Meurice et al. 2016]. Une modélisation de résultats possibles est illustré en figure 3.17. Les liens sont représentés par les arrêtes vertes qui relient les élements des deux sous-graphes ;
4. **analyser la congruence des deux réseaux** telle que formalisée par [Valetto et al. 2007] permettrait d'évaluer l'alignement entre les relations des développeurs au niveau du schéma de la base de données et les relations des développeurs au niveau des accès à la base de données.



(a) Projet A (b) Projet B

FIGURE 3.17 – Exemples de résultats d'analyse de Réseau AccèsDB-SchémaDB

3.6.2 Résultats escomptés

Nous avons décrit deux situations arbitraires. L'une, en figure 3.17a, où la plupart des développeurs d'accès à la base de données ont un lien entre eux en tant que développeurs du schéma de la base de données. L'autre, en figure 3.17b, où tous les développeurs d'accès à la base de données ont un lien vers un seul développeur du schéma de la base de données.

3.6.3 Amélioration possible

Avoir accès aux données historiques de l'évolution du DISS permettrait aussi d'évaluer la congruence au fil du temps. On pourrait imaginer par exemple que la situation pendant un laps de temps du projet soit reflétée par la situation en figure 3.17b et, plus tard, par la situation en figure 3.17a. Ceci refléterait un changement d'organisation de l'équipe.

Pour valider une éventuelle plus-value de cet indice de congruence, il faudrait le corréler à d'autres caractéristiques pour évaluer s'il peut servir de prédicteur.

3.7 Question de recherche 5 : Peut-on détecter des communications formelles entre développeurs qui prouvent que plusieurs développeurs ont collaboré lors du développement du SI ?

Nous ne pouvons pas répondre à cette question car il n'y a pas de données sur les communications formelles décrites dans le schéma conceptuel de [Meurice et al. 2016].

[Bird et al. 2008] ont déja éffectué ce type de travaux et, ensuite, [Wolf et al. 2009] ont affiné ce type de travaux en utilisant les communications en relation avec une tâche. Ils obtiennent les données au moyen de techniques de Mining Software Repositories (MSR)

3.7.1 Données manquantes

Pour apporter une contribution, nous proposons d'enrichir le schéma conceptuel pour permettre la prise en charge de cette information dans le futur. En figure 3.18 nous ajoutons le concept **Communication**. Cet ajout rend possible l'auditabilité des communications lors d'une **Version**. Il est alors possible d'établir un lien entre deux développeurs identifiés par **Sender_id** et **Receiver_id** dans le but de construire le réseau des développeurs ayant communiqués précisement lors de cette version. Une **Communication** peut éventuellement être liée à une tâche dont la nature est décrite dans **TaskDescription**. Si elle n'est pas liée à une tâche, alors il s'agit d'une communication générale entre deux développeurs. Une seule **Task** peut être en rapport avec plusieurs **Communication**. Ceci permet de construire un réseau de communications basé sur une tâche pour cette version ou pour toute les versions si la tâche à été accomplie en plusieurs version.

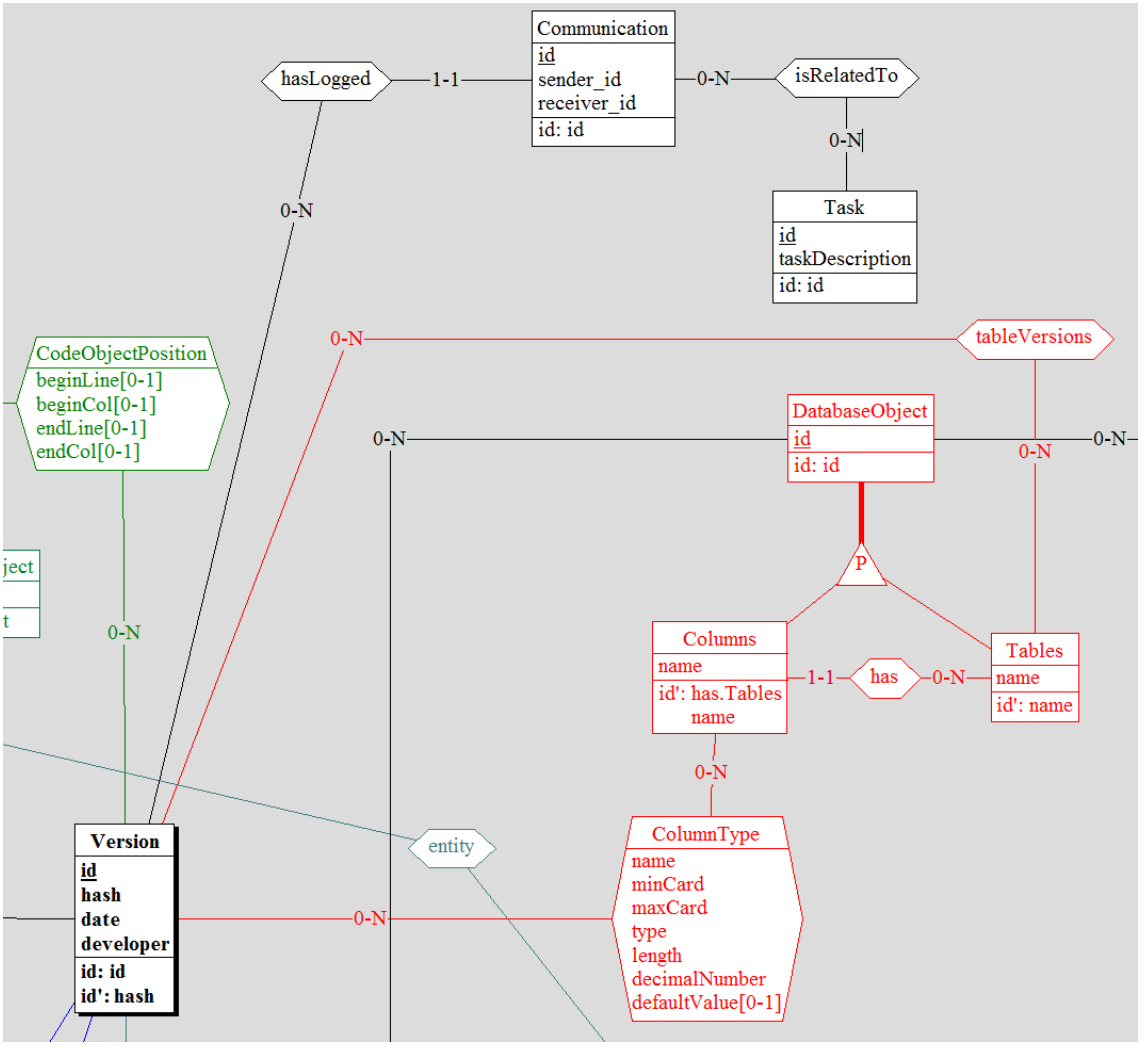


FIGURE 3.18 – Ajout du concept de communication formelle au schéma conceptuel de [MEURICE et al. 2016]

3.7.2 Méthodologie

Nous pouvons généraliser notre approche utilisée en question 1 par les étapes suivantes :

1. **Détecter les communications entre deux développeurs**. Ici, les données de **Communication** et **Task** conceptualisées en figure 3.18.
2. **Etablir un réseau "Développeurs-Communication"** de la manière suivante :
   - dans le cas d'une communication par mail : il s'agirait d'un réseau mixte de noeuds de développeurs et de messages. Le poids des arrêtes entre un développeur et un message sera le nombre de messages envoyés d'un expéditeur à un réceptionnaire ;
   - dans le cas d'une communication autour d'un sujet (e.g. forum de discussion, implémentation d'une tâche, ...) : le poids des arrêtes pourrait être

le nombre de participations à la discussion.

3. **établir un réseau "développeurs-développeurs"** au travers des liens entre les développeurs et les communications qu'ils ont en commun. On établira un lien entre deux développeurs si :
   — dans le cas d'une communication par mail, si un destinataire a répondu au moins une fois à un mail. Il faudrait dés lors filtrer les réponses automatiques comme les out-of-office ;
   — dans le cas d'une communication autour d'un sujet ou d'une tâche en particulier, si ils ont participé ensemble à la même tâche.

Similairement à la question 1, les réseaux obtenus aux étapes 2 et 3 sont alors analysables qualitativement (visualisation des réseaux) ou quantitativement (dérivation d'information supplémentaires a partir des données du réseau comme les noeuds, arrêtes et le poids de celles-ci).

3.7.3 Résultats escomptés

Réseau "Développeurs-Communication"

Un exemple de réseaux développeurs en relation avec des communication par mail est illustré en figure 3.19.

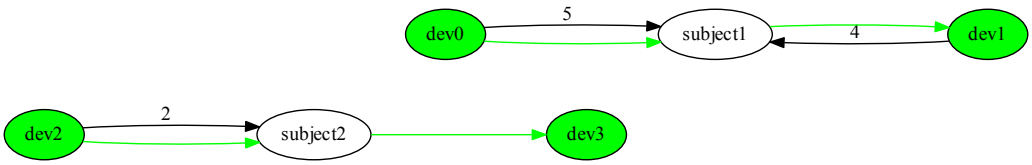


FIGURE 3.19 – Communication email

On constate dans cet exemple que :

— le développeur dev0 a initier un mail vers dev1, ce qui est représenté par les flèches vertes. Cette information provient des attributs **sender_id** et de **receiver_id** modélisé en figure 3.18 ;

— le développeur dev2 a initié un mail vers dev3 ;

— les arrêtes entre les développeurs et les messages existent s'ils ont envoyé ou répondu au **subject**. Le poids de cette arrête dépend du nombre de messages en rapport avec le **subject** ;
   — on constate que dev0 et dev1 ont bien communiqué car ils ont tout les deux une relation avec subject1.

— on constate que dev2 et dev3 n'ont pas bien communiqué car seul dev2 a envoyé 2 messages en relation avec subject2, mais dev3 n'as pas (encore) répondu.

Un exemple de réseaux développeurs en relation avec une tâche est illustré en figure 3.20.

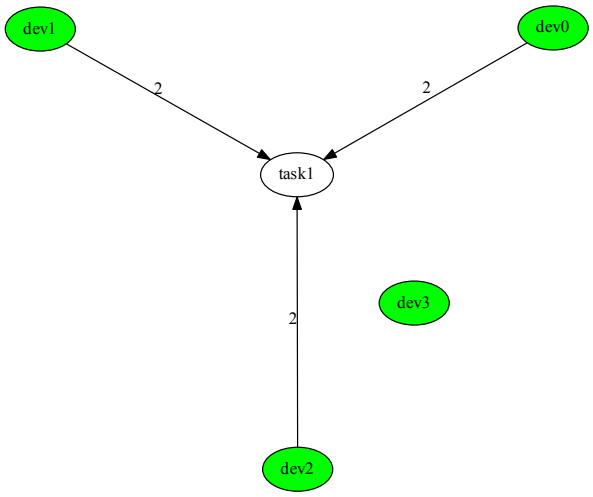


FIGURE 3.20 – Communication autour d'une tâche
On constate dans cet exemple que :

— dev0, dev1 et dev2 communiquent sur la tâche task1.

— dev3 ne communique pas sur la tâche task3.

Réseau "développeurs-Développeurs"

A partir du réseau "Développeurs-Communication" décrit en figure 3.19, nous dérivons le réseau illustré en figure 3.21.

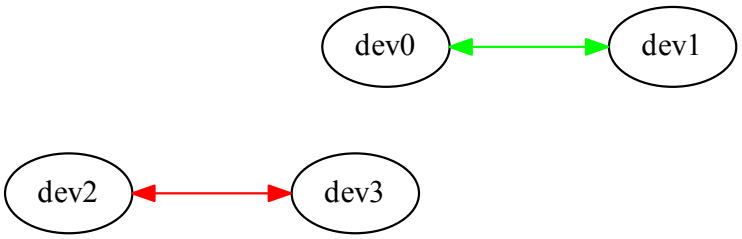


FIGURE 3.21 – Réseau de développeurs en relation au travers d'une communication email
L'arrête verte qui relie dev0 et dev1 représente une correspondance email bidirectionnelle. L'arrête rouge qui relie dev2 et dev3 représente une correspondance email unidirectionnelle et donc un manque de communication.

A partir du réseau "Développeurs-Communication" décrit en figure 3.20, nous dérivons le réseau illustré en figure 3.22.

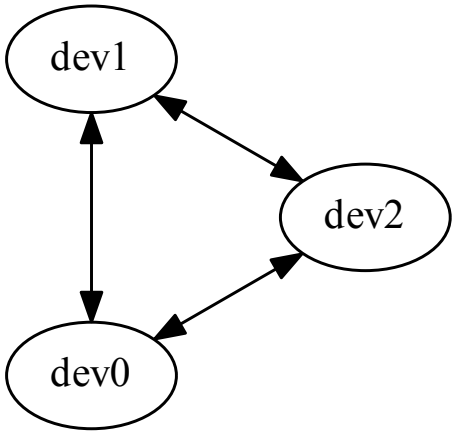


FIGURE 3.22 – Réseau de développeurs en relation au travers d'une tâche commune

3.7.4 Améliorations possible

Nous pensons qu'il serait possible d'améliorer les résultats des questions 1 et 2 en construisant des réseaux qui se limitent aux commits situés entre la date de début et de fin d'une tâche précise. La plus-value, si elle est équivalente aux résultats des travaux de [WOLF et al. 2009], serait de pouvoir identifier les problèmes de collaboration et de communication pour prédire le succès ou l'échec du travail en cours.

A partir des réseaux mixtes "Développeurs-Communication" nous dérivons des réseaux "Développeurs-Développeurs". L'inverse est aussi possible. Il serait en effet envisageable de construire des réseaux "communication-communication" ou plutot "tâches-tâches" qui permettraient de repérer des similarités entre des tâches au travers des développeurs qui y participent. Cette information peut être intéressante pour attribuer une nouvelle tâche à une équipe possédant l'experience nécessaire.

A l'instar de la question 4, il serait intéressant de comparer les réseaux obtenus au travers des artefacts techniques (question 1 ou 2) avec le réseau obtenu au travers des communications formelles (question 5). Ceci permettrait d'évaluer le niveau d'alignement entre les communications formelles et les relations entre les développeurs au travers des artefacts techniques.

3.8 Résumé des résultats obtenus

Question de recherche :	Résumé des résultats
1) Peut-on détecter les développeurs qui ont effectué des changements au schéma de la base de données sur des objets communs (tables, colonnes) ?	Nous sommes capables de construire des réseaux mixtes "développeurs-artefacts techniques" où les artefacts techniques sont soit des tables soit des colonnes. De ces réseaux mixtes, nous dérivons des réseaux "développeurs-développeurs" en utilisant les relations avec un artefact technique commun qui les relient. Nous avons évalué notre méthode sur les données historiques de deux DISS réels.
2) Peut-on détecter les développeurs ayant effectués des changements dans le code du programme sur des appels communs à la base de données ?	Nous montrons que notre approche utilisée lors de la question 1 est généralisable. Elle est donc aussi applicable pour obtenir des réseaux mixtes "développeurs-artefacts techniques" où les artefacts techniques sont des accès aux tables de la base de données. De ces réseaux mixtes seront aussi dérivables des réseaux "développeurs-développeurs".
3) Peut-on déterminer quels développeurs ont été contemporains ?	Grâce à l'historique des versions, nous pouvons établir des liens entre développeurs si leurs soumissions se recouvrent dans le temps. Nous construisons alors un réseau de développeurs où ils sont liés entre eux s'ils sont contemporains. Nous utilisons aussi ce réseau pour augmenter l'information des réseaux obtenus lors de la question 1 pour faire la différence entre des relations de collaboration de façon contemporaine ou des collaborations de turnover. Nous évaluons notre méthode sur des données historiques de deux DISS réels.

Question de recherche :	Résumé des résultats
4) Les comportements collaboratifs repris aux questions 1 et 2 sont-ils semblables ?	Nous décrivons à haut niveau une méthode pour obtenir les réseaux de relations technico-techniques. Nous modélisons aussi le type de résultats que nous aurions voulu atteindre et leur utilité.
5) Peut-on détecter des communications formelles entre développeurs qui prouvent que plusieurs développeurs ont collaboré lors du développement du Système d'information (SI) ?	Par manque de données dans le schéma conceptuel, nous n'avons pas répondre à la question. Néamoins nous proposons une mise à jour du modèle conceptuel pour acceuillir les données nécessaires à représenter les communications formelles. Ensuite, nous montrons que notre approche utilisée lors de la question 1 est généralisable. Elle est donc aussi applicable pour obtenir des réseaux mixtes "développeurs-Communications" où les communications sont des messages mail ou concernant une tâche particulière. De ces réseaux mixtes, seront aussi dérivables des réseaux "développeurs-développeurs".

3.9 Questionnement sur les hypothèses simplificatrices

Comme discuté en section 3.2, nous nous sommes affranchi de certaines considérations. Nous proposons ici des pistes pour les prendre en compte et obtenir des résultats plus précis.

Les données historiques utilisées lors de notre évaluation ne comportent pas un historique de versions jointives.

Les versions absentes peuvent induire un biais car les changements apportés lors des versions absentes peuvent être attribués à un autre développeur que celui qui les a implémentés.

Il faudrait valider notre méthode sur un historique contenant des versions jointives sur toute la branche du projet.

Nous considèrons que les développeurs utilisent toujours un seul identifiant lorsqu'ils travaillent sur le projet.

Il y a une réelle duplication d'identité dans les projets que nous avons évalués et donc nos résultats sont potentiellement biaisés.

— Pour Broadleaf, la table d'identifiants des développeurs en annexe A montre que dev3 (ktisdell@credera.com) , dev10 (ktisdell@broadleafcommerce.org) et dev22 (ktisdell@broadleafcommerce) sont probablement des alias d'une même personne physique. Neanmoins, les alias dev10 et dev22 n'ont pas modifié les colonnes des tables. Il n'y a donc pas de noeud à fusionner avec dev3.

— Pour OpenMRS, la table d'identifiants des développeurs en annexe B montre que dev 28 (kaweesi.joseph2012@gmail.com) est sans doute un alias de dev23 (joseph.kaweesi2012@gmail.com). Néanmoins, l'alias dev23 n'a pas modifié les colonnes des tables. Il n'y a donc pas de noeuds à fusionner avec dev28.

Dans le cas où un développeur effectuerait des modifications en utilisant plusieurs alias, il faudrait fusionner les noeuds représentant le développeur.

Il existe différents algorithmes qui déterminent les identifiants qui représentent une même personne physique [Goeminne et Mens 2013]. Ceux-ci devraient être utilisés pour fusionner les noeuds qui correspondent à la même personne.

3.10 Conclusion

Nous avons apporté une contribution en construisant des réseaux sociaux au travers des modifications apportées à un artefact technique commun de type schéma de base de données. Nous avons aussi évalué notre méthode sur les données historiques de deux systèmes DISS réels et nous avons présenté les résultats obtenus lors de chaque question.

Nous avons observé des différences entre les deux systèmes sur lesquels nous avons validé notre méthode. Lors de l'analyse des changements aux tables nous observons que :

— une proportion plus large des développeurs modifient les tables du schéma dans le cas de Broadleaf (68%) que dans le cas de OpenMRS (9%) ;

— les modifications des tables sont plus courantes dans le cas de Broadleaf que dans le cas d'OpenMRS ;

— les modifications des tables sont parfois sujettes à erreurs dans le cas de Broadleaf.

Lors de l'analyse des changements aux colonnes nous observons que :

— globalement, les changements aux colonnes sont beaucoup plus fréquents que les changements aux tables ;

— pour Broadleaf, la proportion de développeurs modifiant le schéma était toujours plus importante (72%) que celle de OpenMRS (37%) ;

— le nombre de changements par développeur, qui sont apportés aux colonnes, est plus important dans le cas de Broadleaf.

Nous avons aussi discerné les relations entre développeurs au travers des objets qu'ils modifient, selon que ces relations soient contemporaines ou non. Nous constatons alors :

— un phénomène de turnover

— qu'il est possible de mettre en évidence les développeurs qui font le lien tout au long du processus de turnover ;

— que des changements sont apportés tout aussi souvent en l'absence de l'auteur initial qu'en sa présence. Ceci porte à croire que la documentation ou le code servent de moyens de communication indirects ;

— que les réseaux de développeurs sont connexes et ne sont pas déconnectés si on ne tient pas compte des liens non-contemporains. Ceci montre que le turnover est progressif.

Comme annoncé dans notre démarche de questionnement en section 2.2, nous avons porté notre attention sur les aspects collaboratifs au niveau de la maintenance

du schéma de la base de données. Quand les données ne sont pas disponibles, nous proposons de rajouter leur représentation dans le modèle conceptuel. Pour les questions auxquelles nous n'avons pas pu répondre dans le détail, nous décrivons, à haut niveau, une méthode permettant de guider les futures recherches dans ce domaine.

Finalement, nous avons résumé les résultats en section 3.8 et nous discutons en section 3.9 des possibilités permettant de lever les hypothèse simplificatrices.

Conclusion

Il existe des outils permettant d'analyser les processus collaboratifs lors du développement du code d'un SI. De tels outils apportent une réelle plus-value lors de la gestion des efforts d'une équipe. En effet, grâce à ces outils il est possible d'obtenir des informations à propos du degré de coordination d'une équipe[Martinez-Romo et al. 2008]. Il est également possible de comprendre le succès ou l'échec du travail effectué par l'équipe pour accomplir une tâche, et même, de prédire le succès ou l'échec du travail en cours en se basant sur les comportements de collaboration [Wolf et al. 2009]. Cependant, à notre connaissance, les aspects collaboratifs sur la partie du développement du schéma de la base de données n'ont pas été investigués à ce jour au moyen de ces méthodes d'analyse de réseaux sociaux.

Résumé du travail et contributions

Ce mémoire a eu pour but d'explorer les techniques d'analyse de réseaux sociaux dans le domaine des DISS, particulièrement au travers des relations entre les développeurs et les objets qu'ils modifient dans le schéma de la base de données. Nous avons tout d'abord rapellé les concepts nécessaires à la compréhension de ce mémoire. Ensuite, nous avons fait l'état de l'art des méthodes de détection des relations entre les développeurs dans le domaines des SI. Finalement, nous mettons en évidence le manque d'investigation de ces méthodes d'analyse de réseaux sociaux dans le cas du schéma de la base de données et du code y accédant.

Nous apportons notre contribution en analysant les réseaux sociaux que nous avons construits au travers des modifications à un artefact technique commun : le schéma de base de données. Nous avons évalué notre méthode sur les données historiques de deux systèmes DISS réels. Nous en dérivons des réseaux de collaboration entre développeurs. Dans ces réseaux, nous avons fait la différence entre les liens qui existent quand les développeurs sont contemporains ou pas. Nous observons que sur toute la durée de l'historique des projets observés, il y a plus de liens de collaboration entre développeurs quand ils ne sont pas contemporains que lorsqu'ils le sont. Ce qui suggère que le turn-over d'une équipe n'empêche nullement les nouveaux développeurs d'apporter des changements à des artefacts sans avoir à communiquer

directement avec son auteur.

Discussion sur les résultats et limites de la solution proposée

Notre intuition initiale était que même en l'absence de données relatives aux communications formelles entre développeurs, nous pourrions établir des liens de collaboration entre eux au moyen des changements apportés à des artefacts techniques communs. Nous avons construit de tels réseaux en utilisant les changements au schéma de la base de données. Nous avons ensuite établi à l'aide des dates de soumissions des changements qu'une proportion non négligeable des modifications sur des objets communs, que nous considérons comme liens de collaboration, surviennent alors que les développeurs ne sont pas contemporains et donc n'ont pas su communiquer directement. On imagine donc que pour collaborer sur le projet, les développeurs non-contemporains utilisent d'autres canaux de communication en l'absence d'un collaborateur. Nous pensons qu'ils doivent soit utiliser le code et la documentation comme moyen de communication indirecte, soit communiquer formellement avec un autre expert du sujet qui, lui, était le contemporain du développeur original.

Nous avons observé la durée complète du projet, ce qui peut expliquer l'effet du turn-over sur nos résultats. Pour créer une représentation la plus exacte possible d'une équipe travaillant sur l'implémentation d'un changement, il serait utile d'observer une durée plus courte correspondant au temps nécessaire à implémenter un changement. En l'absence de détection de communication formelle, il est encore possible que des contributeurs collaborent sans directement communiquer.

Piste d'amélioration et perspectives

Pour l'instant notre outil fournit des réseaux au format .csv et DOT. Les réseaux au format DOT sont utilisables au moyen d'outils comme GraphViz [1]. Ceci rend l'exploration des résultats fastidieuse pour une éventuelle utilisation régulière. Il serait interessant de fournir un outil qui permettrait d'interagir avec les réseaux de telle manière à offrir les fonctionnalités suivantes :

Fonctionnalités	Utilité
Zoomer sur le noeud d'un développeur, et afficher seulement les artefacts techniques qu'il a modifié.	Reconnaître avec quelle partie du code/schéma le développeur est familier.
Zoomer sur le noeud d'un développeur, et afficher les artefacts techniques qu'il a modifié ainsi que les développeurs ayant modifié les mêmes objects.	Détecter d'autres développeurs qui sont familliers avec la même partie du code/schéma qu'un autre développeur.
Zoomer sur un artefact technique, et afficher les noeuds des développeurs ayant modifiés cet artefact.	Identifier les personnes famillières avec l'artefact technique.
Zoomer sur un artefact technique, afficher les développeurs qui l'ont modifié ainsi que les artefacts techniques que ces développeurs ont modifiés	Identifier d'eventuelles similitudes ou relations entre des artefacts techniques, reverse engineering.
Filtrer les résultats de façon temporelle. Entre un commit de début et un commit de fin.	Limiter la recherche à une phase du développement bien précise.
Permettre de grouper les développeurs au moyen du poids des arrêtes et/ou du nombre d'artefacts techniques avec lesquels les developpeurs sont liés	Détection des structures organisationnelles.
Détecter si la suppression du noeud d'un développeur déconnecte le graphe	Analyse de risque.
Détecter les similarités entre les développeurs selon le nombre d'artefacts techniques qu'ils ont modifiés.	Trouver un remplacant, gestion du risque.
Détecter les développeurs qui sont déconnectés du graphe, c'est-à-dire dont les artefact techniques impactés ne sont jamais modifiés par d'autres développeurs	Détection de specialistes, analyse du risque.

1. `www.graphviz.org`

D'autres pistes d'amélioration seraient de corréler le niveau de collaboration d'une équipe lors de phases du projet connues comme étant un succès ou un echec. Ceci permettrait d'essayer de reconnaître des patterns de collaboration permettant de servir de prédicteur de succès ou d'echec d'un projet.

L'utilité de la plupart de ces fonctionnalités sont d'ordre organisationnel. Dans ce mémoire nous créons des réseaux mixtes "développeurs-artefacts techniques" qui nous permettent de détecter des liens entre les développeurs au travers des artefacts techniques communs. L'inverse est aussi possible, c'est-à-dire de détecter des liens entre les artefacts techniques au moyen des développeurs communs. Ceci peut avoir une utilité pour comprendre la structure du code et du schéma.

Dans le cas du schéma de la base de données, cette hypothèse pourrait être testée pour les projets que nous avions étudiés. Il faudrait d'abord créer les liens entre les tables au moyen des liens entre les développeurs et les tables. Ensuite, il faudrait comparer si les liens obtenus entre les tables correspondent aux clefs étrangères du modèle conceptuel disponible dans la documentation du projet.

Bibliographie

Bird, Christian et al. (2006). « Mining email social networks ». In : *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, p. 137–143.

Bird, Christian et al. (2008). « Latent social structure in open source projects ». In : *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, p. 24–35.

Bolici, Francesco, James Howison et Kevin Crowston (2009). « Coordination without discussion ? Socio-technical congruence and Stigmergy in Free and Open Source Software projects ». In : *Socio-Technical Congruence Workshop in conj Intl Conf on Software Engineering, Vancouver, Canada*.

Briand, Lionel C (2003). « Software documentation : how much is enough ? » In : *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*. IEEE, p. 13–15.

Brooks, Frederick (1995). *The mythical man-month : essays on software engineering*. Reading, Massachusetts : Addison-Wesley Publishing Company. isbn : 0-201-83595-9.

Canfora, Gerardo et al. (2011). « Social interactions around cross-system bug fixings : the case of FreeBSD and OpenBSD ». In : *Proceedings of the 8th working conference on mining software repositories*. ACM, p. 143–152.

CanforaHarman, Gerardo et Massimiliano Di Penta (2007). « New frontiers of reverse engineering ». In : *2007 Future of Software Engineering*. IEEE Computer Society, p. 326–341.

Chapitre 1 : Système d'information de l'entreprise. consulté le 12 août 2017. url : `http://profs.vinci-melun.org/profs/adehors/CoursWeb2/Cours/Ch1/Ch1.php`.

Cleve, Anthony et Jean-Luc Hainaut (2006). « Co-transformations in database applications evolution ». In : *Generative and Transformational Techniques in Software Engineering*. Springer, p. 409–421.

Cleve, Anthony, Tom Mens et Jean-Luc Hainaut (2010). « Data-intensive system evolution ». In : *Computer* 43.8, p. 110–112.

Cleve, Anthony et al. (2015). « Understanding database schema evolution : A case study ». In : *Science of Computer Programming* 97, p. 113–121.

DB-Engines Ranking. consulté le 19 mars 2017. url : http://db-engines.com/en/ranking.

De Souza, Cleidson, Jon Froehlich et Paul Dourish (2005). « Seeking the source : software source code as a social and technical artifact ». In : *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*. ACM, p. 197–206.

Email de Loup Meurice, UNamur faculté d'informatique. 18 décembre 2015. pers. comm.

Gacek, Cristina et Budi Arief (2004). « The many meanings of open source ». In : *IEEE software* 21.1, p. 34–40.

Goeminne, Mathieu, Alexandre Decan et Tom Mens (2014). « Co-evolving code-related and database-related changes in a data-intensive software system ». In : *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, p. 353–357.

Goeminne, Mathieu et Tom Mens (2013). « A comparison of identity merge algorithms for software repositories ». In : *Science of Computer Programming* 78.8, p. 971–986.

guide, PMBOK (2013). *A guide to the project management body of knowledge (PMBOK guide)*. Newtown Square, Pennsylvania : Project Management Institute, Inc. isbn : 978-1-935589-67-9.

Hainaut, Jean-Luc (2009). *Bases de données : Concepts, utilisation et développement*. Dunod.

IBM Rational Team Concert. consulté le 6 août 2017. url : https://jazz.net/products/rational-team-concert/.

Martinez-Romo, Juan et al. (2008). « Using social network analysis techniques to study collaboration between a FLOSS community and a company ». In : *IFIP International Conference on Open Source Systems*. Springer, p. 171–186.

Mens, Tom et Mathieu Goeminne (2011). « Analysing the evolution of social aspects of open source software ecosystems. » In : *IWSECO@ ICSOB*, p. 1–14.

Meurice, Loup et Anthony Cleve (2014). « Dahlia : A visual analyzer of database schema evolution ». In : *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, p. 464–468.

— (2016). « DAHLIA 2.0 : A Visual Analyzer of Database Usage in Dynamic and Heterogeneous Systems ». In : *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*. IEEE, p. 76–80.

Meurice, Loup et al. (2016). « Analyzing the Evolution of Database Usage in Data-Intensive Software Systems ». In :

Nakakoji, Kumiyo et al. (2002). « Evolution patterns of open-source software systems and communities ». In : *Proceedings of the international workshop on Principles of software evolution*. ACM, p. 76–85.

Ohira, Masao et al. (2005). « Accelerating cross-project knowledge collaboration using collaborative filtering and social networks ». In : *ACM SIGSOFT Software Engineering Notes*. T. 30. 4. ACM, p. 1–5.

OpenMRS Data Model 1.9.0. consulté le 12 août 2017. URL : `https://wiki.openmrs.org/display/docs/Data+Model?preview=/589829/34374263/openmrs_data_model_1.9.0.png`.

Sarma, Anita et al. (2009). « Tesseract : Interactive visual exploration of socio-technical relationships in software development ». In : *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, p. 23–33.

Sureka, Ashish, Atul Goyal et Ayushi Rastogi (2011). « Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis ». In : *Proceedings of the 4th india software engineering conference*. ACM, p. 195–204.

Valetto, Giuseppe et al. (2007). « Using software repositories to investigate socio-technical congruence in development projects ». In : *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE, p. 25–25.

Van Antwerp, Matthew et Greg Madey (2010). « The importance of social network structure in the open source software developer community ». In : *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE, p. 1–10.

Version Control Systems Popularity in 2016 (2016). consulté le 4 août 2017. URL : `https://rhodecode.com/insights/version-control-systems-2016`.

What is version control | Atlassian Git Tutorial. consulté le 4 août 2017. URL : `https://www.atlassian.com/git/tutorials/what-is-version-control`.

Wolf, Timo et al. (2009). « Mining task-based social networks to explore collaboration in software teams ». In : *IEEE Software* 26.1, p. 58–66.

Annexe A

Liste des développeurs du projet Broadleaf

id	developer
dev0	jball@credera.com
dev1	rsmith@credera.com
dev2	btaylor@credera.com
dev3	ktisdell@credera.com
dev4	jfischer@broadleafcommerce.org
dev5	aangus@credera.com
dev6	mchou@credera.com
dev7	jfridye@credera.com
dev8	dtalk@credera.com
dev9	bpolster@broadleafcommerce.org
dev10	ktisdell@broadleafcommerce.org
dev11	ebautista@broadleafcommerce.org
dev12	pverheyden@gmail.com
dev13	apazzolini@gmail.com
dev14	ppatel@broadleafcommerce.org
dev15	jfridye@gmail.com
dev16	jerry77oz@gmail.com
dev17	t_leffert@yahoo.com
dev18	marshall.treadaway@gmail.com
dev19	jlutteringer@credera.com
dev20	jon.fleschler@gmail.com
dev21	arooke@broadleafcommerce.com
dev22	ktisdell@broadleafcommerce.com
dev23	bsmith@broadleafcommerce.com
dev24	pbaggett@Philips-MacBook-Pro.local

Annexe B

Liste des développeurs du projet OpenMRS

id	developer
dev0	ben@openmrs.org
dev1	jazayeri@alum.mit.edu
dev2	bmckown@regenstrief.org
dev3	jcm62@columbia.edu
dev4	vergil@gmail.com
dev5	pihdave@gmail.com
dev6	sunbiz@gmail.com
dev7	akollegger@gmail.com
dev8	robby.oconnor@gmail.com
dev9	nyoman@openmrs.org
dev10	mseaton@pih.org
dev11	syhaas@gmail.com
dev12	wyclif@openmrs.org
dev13	hablutzel1@gmail.com
dev14	dkayiwa@openmrs.org
dev15	rafal@openmrs.org
dev16	kishoreyekkanti@gmail.com
dev17	jeremy@openmrs.org
dev18	jriley@runningwave.net
dev19	benwolfe@gmail.com
dev20	rafal.korytkowski@gmail.com
dev21	lluismf@gmail.com
dev22	wyclif@openmrs.com
dev23	joseph.kaweesi2012@gmail.com

id	developer
dev24	rowanseymour@gmail.com
dev25	arkadiusz.kolodziejski@gmail.com
dev26	bhashitha427@gmail.com
dev27	mikepigg@highlylogical.com
dev28	kaweesi.joseph2012@gmail.com
dev29	filip.spiridonov@gmail.com
dev30	chalakanth@seonthemon.com
dev31	marv@hostin.is
dev32	dkithmalfit@gmail.com
dev33	akash.wanted@gmail.com
dev34	mgoodrich@pih.org
dev35	citigo@ehs.cl
dev36	endeep123@gmail.com
dev37	harsz89@gmail.com
dev38	chaityabshah@verizon.net
dev39	kmit.satwikreddy@gmail.com
dev40	itatriev@gmail.com
dev41	tmueller@soldevelo.com
dev42	jo.adam.93@gmail.com